UNIVERSIDADE DE LISBOA
Faculdade de Ciências
Departamento de Informática

NOVELTY GRAMMAR SWARMS

Diana Filipa Guerreiro Galvão

DISSERTAÇÃO

MESTRADO EM ENGENHARIA INFORMÁTICA
Especialização em Sistemas de Informação

Dissertação orientada pelo Prof. Doutor Paulo Jorge Cunha Vaz Dias Urbano
e co-orientada pelo Prof. Doutor Joel Lehman

2015

Acknowledgments

First I would like to thank my supervisor, Professor Paulo Urbano, for guiding my work through the year and always being available to help and providing me with new ideas to improve my work. I would like to thank my co-supervisor Joel Lehman for, despite the distance and tight schedule, always being able to help in every way possible when needed. Also, I would like to thank Fundação para a Ciência e Tecnologia for making my work possible by providing the financial support. Thanks to my parents for their unconditional support even in difficult times and even if they did not quite understood what I was doing. I would also like to thank my aunt Paula and cousin Sofia which are always fun to be around. Finally, I would like to thank to André Lamúrias for all the help, motivation and inspiration needed to accomplish my goals.

To my parents.

Resumo

Particle Swarm Optimization (PSO) é um dos métodos de optimização populacionais mais conhecido. Normalmente é aplicado na otimização funções de *fitness*, que indicam o quão perto o algoritmo está de atingir o objectivo da pesquisa, fazendo com que esta se foque em áreas de *fitness* mais elevado. Em problemas com muitos ótimos locais, regularmente a pesquisa fica presa em locais com *fitness* elevado mas que não são o verdadeiro objetivo. Com vista a solucionar este problema em certos domínios, nesta tese é introduzido o Novelty-driven Particle Swarm Optimization (NdPSO). Este algoritmo é inspirado na pesquisa pela novidade (*novelty search*), um método relativamente recente que guia a pesquisa de forma a encontrar instâncias significativamente diferentes das anteriores. Desta forma, o NdPSO ignora por completo o objetivo perseguindo apenas a novidade, isto torna-o menos susceptivel a ser enganado em problemas com muitos optimos locais. Uma vez que o *novelty search* mostrou potencial a resolver tarefas no âmbito da programação genética, em particular na evolução gramatical, neste projeto o NdPSO é usado como uma extensão do metodo de *Grammatical Swarm* que é uma combinação do PSO com a programação genética.

A implementação do NdPSO é testada em três domínios diferentes, representativos daqueles para o qual este algoritmo poderá ser mais vantajoso que os algoritmos guiados pelo objectivo. Isto é, domínios enganadores nos quais seja relativamente intuitivo descrever um comportamento. Em cada um dos domínios testados, o NdPSO supera o aloritmo standard do PSO, uma das suas variantes mais conhecidas (Barebones PSO) e a pesquisa aleatória, mostrando ser uma ferramenta promissora para resolver problemas enganadores.

Uma vez que esta é a primeira aplicação da pesquisa por novidade fora do paradigma evolucionário, neste projecto é também efectuado um estudo comparativo do novo algoritmo com a forma mais comum de usar a pesquisa pela novidade (na forma de algoritmo evolucionário).

Palavras-chave: Otimização por Enxames de Particulas, Pesquisa pela novidade, Paorgramação Genética, Evolução grammatical, Enxames Gramaticais, Problemas enganadores

Abstract

Particle Swarm Optimization (PSO) is a well-known population-based optimization algorithm. Most often it is applied to optimize fitness functions that specify the goal of reaching a desired objective or behavior. As a result, search focuses on higher-fitness areas. In problems with many local optima, search often becomes stuck, and thus can fail to find the intended objective. To remedy this problem in certain kinds of domains, this thesis introduces Novelty-driven Particle Swarm Optimization (NdPSO). Taking motivation from the novelty search algorithm in evolutionary computation, in this method search is driven only towards finding instances significantly different from those found before. In this way, NdPSO completely ignores the objective in its pursuit of novelty, making it less susceptible to deception and local optima. Because novelty search has previously shown potential for solving tasks in Genetic Programming, particularly, in Grammatical Evolution, this paper implements NdPSO as an extension of the Grammatical Swarm method which in effect is a combination of PSO and Genetic Programming.

The resulting NdPSO implementation was tested in three different domains representative of those in which it might provide advantage over objective-driven PSO, in particular, those which are deceptive and in which a meaningful high-level description of novel behavior is easy to derive. In each of the tested domains NdPSO outperforms both objective-based PSO and random-search, demonstrating its promise as a tool for solving deceptive problems.

Since this is the first application of the search for novelty outside the evolutionary paradigm an empirical comparative study of the new algorithm to a standard novelty search Evolutionary Algorithm is performed.

Keywords: Particle Swarm Optimization, Novelty Search, Genetic Programming, Grammatical Evolution, Grammatical Swarm, Deceptive problems

Resumo Alargado

Particle Swarm Optimization (PSO), introduzido em 1995 por Kennedy e Eberhard [1], é, até há data, um dos metodos de optimização de funções baseado em populações de indivíduos mais usado. No ínicio dos anos 90, a forma como certas espécies de animais se comportam entre si, por vezes chamado comportamento social, era alvo de muitos estudos nas mais diversas áreas. Estes estudos, principalmente aqueles que se focavam no movimento em bando dos pássaros, foram a inspiração dos autores do PSO. Na sua essência, o PSO é um conceito simples inspirado em eventos biológicos. A sua simplicidade torna-o um dos metodos computacionais mais simples de compreender.

No algoritmo do PSO, a população, ou exame, é composta por partículas que se movem no espaço multi-dimensional com a finalidade de optimizar uma função de fitness. Esta função indica o quão perto a pesquisa está do seu objetivo final. A cada partícula está associado um vetor de posição e outro com a sua velocidade, ambos atribuidos de forma aleatória no ínicio do processo de otimização. Para além destes vetores, cada partícula tem também um valor de fitness que depende da sua posição e corresponde ao resultado da função de fitness.

Adicionalmente, as partículas têm também uma componente de memória que guarda a sua experiência passada. Em particular, cada partícula armazena a posição onde obteve melhor fitness, *personal-best* ou *pbest*. As partículas também partilham informação entre si, armazenando a posição que gerou o melhor fitness de todo o exame, chamada de *global-best* ou *gbest*. Estas componentes ajudam a manter o balanço entre uma exploração mais grosseira de todo o espaço de resultados e uma exploração mais detalhada de algumas áreas mais específicas do mesmo [2]. Na prática, a comunicação entre partículas pode ser condicionada usando diferentes topologias de vizinhança, neste caso, o *gbest* será local a cada vizinhaça.

Ainda que bastante popular e eficiente em muitos casos, como muitos outros métodos baseados em populações de partículas, em problemas mais complexos e enganadores, o PSO é muito suscetivel a convergência permatura para ótimos locais [3, 4]. Como foi mencionado anteriormente, maioria das aplicações do PSO otimizam uma função de fitness que estima o progresso em relação ao objetivo pretendido. Guiar a pesquisa em função do objetivo, faz com que a mesma se foque em áreas de maior fitness, ignorando áreas de fitness menor, e, como consequência, algumas partes do espaço de resultados não

serão exploradas. Em problemas mais simples este processo é bastante eficiente, mas em problemas enganadores pode tornar-se problemático. Isto é, em certos problemas pode acontecer que o objetivo se encontre para lá de uma área com fitness mais baixo. A não exploração dessas áreas pode fazer com que o algoritmo nunca chegue a atingir o objetivo pretendido.

A convergência prematura para ótimos locais no PSO é um problema bem conhecido entre os investigadores. Ao longo dos anos muitas variações com vista a combater este problema têm vindo a ser propostas [5, 6]. Mesmo que algumas consigam superar a performance do algoritmo standard do PSO em domínios pouco enganadores, umas vez que estas continuam a ser guiadas pela distância ao objetivo, continuam a ser suscetiveis a convergência prematura se a função for suficientemente enganadora. Desta forma, existe uma relação clara entre pesquisa guiada pelo objectivo e convergência prematura. Assim, para evitar que a pesquisa convirja prematuramente em problemas enganadores, é necessário que não se considere, de todo, o objetivo.

A pesquisa pela novidade, ou *Novelty Search*, é um Algoritmo Evolucionário (EA) que dá este passo radical [7] e, apesar de relativamente recente, já foi aplicado com sucesso em diferentes áreas, como a neuroevolução [7, 8] e a Programação Genética (GP) [9, 10]. A principal motivação deste método é de que a novidade (neste caso, diferenças relevantes entre o comportamento de um individuo em relação aos outros) é uma fonte de informação valiosa. Assim, em vez de guiar a pesquisa estimando a distância ao objetivo, a pesquisa pela novidade é guiada para as instâncias significativamente diferentes daquelas encontradas anteriormente. Por outras palavras, em vez de se medir o quão perto cada individuo está do objetivo, é aplicada uma medida de novidade que mede o quão diferente é o comportamento de cada individuo em relação outros encontrados até então.

Na área evolucionária têm sido propostas outras técnicas que promovem diversidade genética com o objetivo de prevenir a convergência prematura em algoritmos guiados pelo objetivo [11, 12, 13]. Contudo, manter a novidade entre comportamentos, como na pesquisa pela novidade, tem provado proporcionar melhores resultados a superar a convergência prematura [14]. Note-se que na pesquisa pela novidade apenas se procura por comportamentos diferentes, ignorando o objetivo por completo. Desta forma, para se poder aplicar a pesquisa pela novidade a um determinado domínio, é necessário a criacão de um descritor de comportamentos significante para esse domínio. Por exemplo, numa simulação num labirinto, o descritor do comportamento pode ser as coordenadas [x,y] finais do indivíduo depois de esgotar o limite de passos ou de chegar ao objetivo. Na mesma simulação, outro descritor de comportamento poderia ser as coordenadas [x,y] ao longo do tempo, considerando um intervalo de amostragem.

O metodo proposto nesta tese combina, pela primeira vez, o PSO com a pesquisa pela novidade, de forma a combater o problema da convêrgencia prematura para ótimos locais no PSO. Este método foi batizado de Novelty-driven Particle Swarm Optimiza-

tion (Novelty-driven PSO ou NdPSO). Em trabalhos anteriores a pesquisa pela novidade demostrou proporcionar bons resultados quando aplicada juntamente com Programação Genética, mais própriamente, Evolução Gramatical (GE) [15]. Com base nestes resultados, aqui, o NdPSO é aplicado como uma extensão do Grammatical Swarm (GS), ou Enxames de Gramáticas [16], esta implementação foi chamada de Novelty-driven Grammatical Swarm (NdGS). GS é um método que junta o PSO com a Evolução Gramatical. Por sua vez, esta última, aplica o processo de mapeamento usado em Programação Genética com o uso de gramáticas de forma a conseguir produzir programas compiláveis em qualquer linguagem de programação.

O metodo proposto foi testado em três domínios respresentativos daqueles para os quais o NdPSO possa ser mais apropriado. Estes domínios, devem ser enganadores (caso contrário a pesquisa baseada em objectivo é mais eficiente) e proporcionar uma forma intuitiva de caracterizar um espaço de comportamentos relevantes para o domínio (caso contrário é dificil aplicar a pesquisa pela novidade). Primeiro o NdPSO foi aplicado no problema do Mastermind, tal como no jogo no qual é inspirado, neste domínio o algoritmo tenta descobrir uma sequencia de pins escondida. A forma como a função de fitness foi definida torna este problema bastante enganador e desafiante para o PSO. Os outros dois domínios (o Trilho de Santa Fe e o Problema de Navegação num Labitinto) são benchmarks de aprendizagem por reforço usados em Programação Genética por serem bastante enganadores. Estas experiências comparam a performance do NdPSO com a pesquisa aleatória e o PSO guiado pelo objectivo (incluido uma das variantes mais conhecidas do algoritmo standard - Barebones PSO). Em todos os domínios testados o NdPSO tem uma performance bastante melhor que os metodos guiados por objectivo e que a pesquisa aleatória, revelando o seu potencial para a resolução de problemas enganadores.

Para além de comparar o metodo desenvolvido com o PSO orientado por objectivo é tambem importante verificar se aquele é competitivo com a implementação standard da pesquisa pela novidade, isto é, aplicado em Algoritmos Genéticos. Assim, nesta tese tambem é efectuado um estudo empírico entre o NdPSO e a pesquisa pela novidade standard. Uma vez que o NdPSO foi implementado usando a Evolução Gramatical, de forma a fazer uma comparação justa, também a pesquisa pela novidade é implementada com o algoritmo GE e nos mesmos domínios de bechmark usados anteriormente (Mastermind, Santa Fe Ant Trail e o Problema de Navegação num Labirinto).

Quando comparado com a implementação evolucionaria da pesquisa pela novidade, o NdPSO foi capaz de a superar num dos três problemas testados, o Mastermind. Apesar de ficar abaixo da performance nos outros dois, o NdPSO tem bastante espaço para ser melhorado no futuro. Com menos parâmetros que os algoritmos genéticos, sendo mais intuitivo e fácil de implementar, o NdPSO mostra ter bastante potencial.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Particle Swarm Optimization (PSO) is an effective, general, and popular population-based optimization method [1]. Its creators, James Kennedy and Russell Eberhard, were inspired by animal social behavior, and more specially, by the flocking behavior of birds. In essence, PSO is conceptually simple to understand based on its biological inspiration, in particular on the swarming behavior of insects and specially on the flocking behavior of birds. In nature it is common to observe many species of animals that aggregate together coordinating their actions to perform tasks. By profiting from discoveries and previous experience of other members, this type of behavior allows those animals to accomplish tasks and solve problems beyond the capabilities of a single individual [17]. As those animals move for many different reasons such as to avoid predators, in modeling animal social behavior the concept of changing positions is relatively simple. That is, in the simulation, one agent, or particle (as it is often called in the PSO context) changes its position and evaluates it. Then, its next step is based on this evaluation.

Although PSO is a popular and effective algorithm, like other population-based methods it is susceptible to converge prematurely to *local optima* when applied to complex or deceptive problems [3, 4]. Most applications of PSO optimize an objective-based fitness function that estimates progress towards the desired outcome, e.g. minimizing squared error or maximizing similarity to a goal behavior. Attempting to guide the search directly towards its ultimate goal increases focus on higher-fitness areas at the expense of lower-fitness ones, reducing overall exploration of the search space. In simple problems this focus aids efficiency, but in deceptive problems it is problematic. That is, the pervasiveness of *local optima* may render the objective reachable only by traveling across areas with low objective-based fitness. By pruning away such low-fitness areas, an algorithm may paradoxically preclude itself from reaching its objective.

Because local optima are a general and well-known issue in search (and therefore in PSO), many researchers have proposed variations of PSO to circumvent premature convergence [5, 6]. While such variations may outperform the standard PSO algorithm in domains with limited deception, because they remain guided by the objective, they are still

vulnerable to premature convergence if the objective function is sufficiently deceptive. In this way, there is a clear relationship between objective-based search, deception, and premature convergence. Thus to avoid premature convergence in very deceptive domains it may be necessary to guide search *without* considering the ultimate objective at all.

Novelty search (NS) is an Evolutionary Algorithm (EA) which takes this radical step [7], and has been successfully applied in neuroevolution [7, 8] and Genetic Programming (GP) [9, 10], but not yet to PSO, which is a main contribution of this thesis. The core insight motivating novelty search is that novelty, i.e. demonstrating qualitative difference from previously encountered individuals, is a valuable source of information. Thus, instead of guiding search by estimated distance to the search's objective, novelty search is instead driven towards instances significantly *different* from those found before. In other words, instead of measuring how close an individual is to the objective(i.e. an objective-based fitness function), a quantitative measure of novelty is applied to measure how different an individual's behavior is from others that have been found so far. Note that there is a distinction between the genome search space, which is usually high-dimensional, and the behavior space, which is where the Novelty Search algorithm searches. Behavior Space is usually low-dimensional by design and it is a user-defined description of behaviors. For example, a behavioral description of a maze navigation robot might ignore the actions and trajectory of the robot and characterize it just by its final location. In this way, to apply novelty search to a new domain requires the experimenter to devise a characterization of behavior in that domain. While other successful techniques have been proposed aiming to prevent premature convergence by promoting *genotypic* diversity [11, 12, 13], promoting *behavioral* diversity (as novelty search does), often provides better results [14] because many different genotypes can map to the same underlying behavior.

Motivated by the effectiveness of both novelty search and PSO, this thesis introduces Novelty-driven PSO (NdPSO), as a way to combat premature convergence when applying PSO to deceptive problems. Because novelty search has shown prior promise in combination with GP [15], here NdPSO is implemented as an extension of the Grammatical Swarm (GS) method [16], a PSO-based version of a popular GP algorithm. This particular implementation is thus called Novelty-driven Grammatical Swarm (NdGS).

1.1 Motivation

Since its inception in 1995, Particle Swarm Optimization has become one of the most well-known population-based optimization algorithms. Although in many problems it can provide good and fast results, in others the search prematurely converges, precluding finding the objective of search; this problem is known as *premature convergence*. Most applications of PSO optimize an objective-based fitness function which estimates the progress to the desired outcome. However, guiding the search directly towards the ul-

timate goal causes it to focus on higher-fitness areas at the expense of lower-fitness ones, reducing the overall exploration of the search space.

Because premature convergence is a well-known issue in PSO, many researchers have proposed variations to the standard algorithm to circumvent this problem. The majority of these variations remain guided by heuristic distance to the objective, making them still vulnerable to premature convergence if the objective function is sufficiently deceptive. Because the relationship between objective-based search and premature convergence is clear, to avoid it, especially in very deceptive domains, it may be necessary to guide the search without considering the ultimate objective at all.

Inspired by the success of novelty search, this thesis presets a new method that couples novelty search with PSO. Novelty search is an Evolutionary Algorithm that guides search only towards instances significantly different from those found before. However, to this date, NS has only been applied to an evolutionary optimization context. In this new method the core PSO algorithm is maintained, but instead of measuring how close a particle is to the objective (the standard approach), a novelty metric is applied to measure how different a particle's behavior is from others that have been found so far, and search attempts to maximize such novelty.
By taking the radical step of ignoring the objective completely, this method circumvents the problem of deception inherent in objective-based algorithms such as PSO.

1.2 Objectives

The following represents the thesis hypothesis.

Hypothesis: By ignoring the objective and guiding the search only towards novel behaviors the issue of premature convergence in Particle Swarm Optimization can be avoided.

In order to fulfill the thesis hypothesis, the following objectives were outlined:

- To develop a new method - Novelty-driven Particle Swarm Optimization (NdPSO), which combats the challenge of premature convergence to local optima in PSO. Building on previous successes of novelty search in evolutionary algorithms, this new method applies novelty search to PSO.

- To evaluate the performance of the new method by comparing it to the standard PSO algorithm and a PSO variation (Barebones PSO) in domains representative of those for which the algorithm is most appropriate. In particular, such domains are deceptive (otherwise an objective-based search method is likely to be more effective) and provide an intuitive way to characterize a space of behaviors capturing important aspects of the domain (otherwise it is difficult to apply novelty search).

- To verify if the search for novelty can be successfully transferred to population-based optimization algorithms outside the evolutionary paradigm. Therefore, an empirical study is performed comparing the new algorithm to a standard novelty search Evolutionary Algorithm in the same domain.

1.3 Contributions

The following are the main contributions of this thesis:

- The introduction of Novelty-driven Particle Swarm Optimization (Novelty-driven PSO or NdPSO), a new method to combat the challenge of premature convergence in PSO. In NdPSO instead of measuring and minimizing how close a particle is to the objective, a novelty metric is applied and maximized, to measure and encourage how different an particle's behavior is from others that have been found so far. To the author's knowledge, this is the first application of novelty search outside of evolutionary algorithms.
- A particular implementation of Novelty-driven PSO using Grammatical Swarm, called Novelty-driven Grammatical Swarm (NdGS). GS is a generalization of the classical PSO which searches through a space of programs (in an arbitrary programming language) using a context-free grammar.
- The choice of Barebones PSO as the variation of PSO that aims at preventing premature convergence and the implementation of a grammatical adaptation of Barebones PSO.
- Application of this method to the Mastermind problem, the Santa Fe Ant Trail, and Maze Navigation. The aim is to compare the performance of this algorithm to the traditional objective-driven GS.
- Verification of the competiveness of this PSO-based algorithm with the standard evolutionary novelty-search alogirthm. Therefore I have performed a study comparing NdGS with standard novelty search combined with Grammatical Evolution over the same three benchmarks.

This work motivated a paper that was submitted and accepted for the Biennial International Conference on Artificial Evolution (EA-2015) that will take place from 26 to 28 October 2015 in Lyon, France.

1.4 Structure of the document

This document is organised as follows:

- Chapter 2: *"Related Work"* - Provides a review of the backround literature related to this thesis. First, Particle Swarm Optimization is reviewed, followed by the key concepts of Evolutionary Computation and then Grammatical Evolution (GE). Then the Grammatical Swarm (GS) method is presented, and the chapter concludes with a review of Novelty Search.

- Chapter 3: *"Novelty-driven Particle Swarm Optimization"* - A description of Novelty-driven Particle Swarm Optimization, the method proposed in this thesis, is provided. NdPSO builds from the background provided in the previous chapter to overcome deception in PSO. Then this method is applied to three well-known problem domains and compared to objective-driven Standard PSO, Barebones PSO and to random search. The results are presented and then discussed in the end of this chapter.

- Chapter 4: *"Novelty-driven PSO and Novelty Search Comparative Study"* - An empirical comparative study of the developed method NdPSO and the standard implementation of novelty search (as an Evolutionary Algorithm) is presented in this chapter. After an introduction about the goals of this study, a theoretical comparison between Particle Swarm optimization and a generic Genetic Algorithm is made. The chapter ends with a discussion about the results obtained in the experiments developed followed by the conclusion.

- Chapter 5: *"Conclusion"* The main conclusions of this work are discussed and some directions for future work are indicated.

Chapter 2

Related Work

This chapter presents the background work that this thesis extends upon. It provides context to the Novelty-driven Particle Swarm Optimization algorithm presented in the following chapter. First, Particle Swarm Optimization is reviewed, including popular variations of the standard algorithm. Next, the key concepts of Evolutionary Computation (EC) including Genetic Programming are introduced followed by a review of Grammatical Evolution and its mapping process. Afterwords, Grammatical Swarm, a fairly recent method that combines PSO and the GE mapping process is presented. This chapter concludes with a review of Novelty Search, a recent method originally introduced in Evolutionary Computation, that aims to overcome premature convergence in deceptive problems.

2.1 Particle Swarm Optimization

The term *swarm intelligence*, introduced by Gerardo Beni and Jing Wang, characterizes the distributed intelligence of agents who aggregate and coordinate their actions in order to perform certain tasks [18]. From finding food more efficiently or protecting themselves from predators, this type of behavior is common in many species of animals. Through enabling agents to profit from the discoveries and previous experiences of their neighbors, social behavior facilitates accomplishing tasks beyond the capabilities of any single individual [17].

For example, ants acting together can find the shortest path to a food source, defend their colony from neighbors, and even efficiently allocate different types of workers to perform different tasks. Birds provide another example of animals that benefit from swarm intelligence. They flock together for many different reasons, one of which is protection. That is, a larger group of birds has a better chance of spotting and evading a predator through mobbing or agile flight. Another reason is efficient transportation, as birds often arrange themselves in flocks with specific shapes to take advantage of changing wind patterns, allowing them to travel in an energy efficient way.

Sharing information gives some animals an evolutionary advantage. That is one rea-

son why scientists create computer simulations of their interpretation of social behavior. Their goal is to discover the underlying rules behind this type of behavior and replicate swarms and flocks as close as they can. The bird flocking models of Reynolds [19] and Heppner and Grenander [20] are canonical examples. They relate the synchrony of flocking with birds' efforts to maintain an optimal distance from their neighbors. This way, each bird's behaviors affects the that of other birds, pulling them to the right position in the flock.

Based on these previous experiments Kennedy and Eberhard developed a new optimization algorithm model called Particle Swarm Optimization [1]. PSO is a simple algorithm and is easy to comprehend from analogy with its biological inspiration. To date PSO still is one of the most popular population-based methods for solving optimization problems. It has proven to provide good and fast results in many problems, often comparable in performance with more traditional Genetic Algorithms (GA).

This section reviews the main concepts of PSO, a population-based optimization algorithm.

2.1.1 The Standard Particle Swarm Optimization Algorithm

In PSO, the population (or swarm) is composed of particles that move through a $\mathbb{R}^d$ search space, optimizing a fitness function with the following domain $f : \mathbb{R}^d \to \mathbb{R}$, with *d* representing the dimensionality of the search space. Each particle $i \in (1, 2, 3, ..., N)$ is associated with two *d*-dimensional vectors, one recording its position $x_i = (x_{i1}, x_{i2}, x_{i3}, ..., x_{id})$ and the other its velocity $v_i = (v_{i1}, v_{i2}, v_{i3}, ..., v_{id})$, both randomly initialized when the algorithm begins. Each particle has a fitness value, which is the outcome of the fitness function evaluated at the particle's current position. Note that the particle's current position reflects a possible setting of the parameters of the optimization problem, i.e. a potential solution to the problem.

Additionally, particles have a simple memory component to store previous experience. In particular, each particle records the position where it has so far encountered the highest fitness score, which is called its personal-best or $pbest$. Particles also can share information with each other, and also record the point in the search space where the overall best fitness has been obtained among the collective of particles, which is called its global-best $gbest$. These components help balance exploiting promising areas with exploring more broadly [2]. In practice, communication between particles is often restricted by use of a neighborhood topology, meaning that a particle's *gbest* may be calculated from the best search locations recorded by its neighboring particles [21]. Common neighborhood topologies in PSO are discussed later in section 2.1.3.

The PSO algorithm starts by creating the initial population. In this step, N particles are created with random initial positions and velocities. Later studies suggest that some methods which control the initial population can increase the performance of the algo-

rithm [22, 23]. However, this section will focus on the standard version of PSO (from now on in this document this version will be referred to as Standard PSO). Next, at every time-step each particle updates its velocity and calculates its new position. The velocity update is affected by the particle's previous velocity, the *pbest* position, the *gbest* position, the inertia ω and other random parameters described later. Consequently, the particle's new position depends on its own past and on other particles in the swarm.

The Algorithm 1 shows the pseudo-code containing the steps in the standard PSO algorithm.

Algorithm 1 Stanndard PSO algorithm

```
1:  procedure PSO–PROCEDURE
2:      % Create population with random positions and velocities:
3:      cratePopulation                    ▷ % Assign random positions and velocities
4:      for each Particle do
5:          evaluatePopulation                          ▷ % Assign particle's fitness
6:          updatePbest                  ▷ % Set particle's pbest its current position
7:      end for
8:      updateGbest
9:      % Optimization Process:
10:     while CriteriaAreNotMet do
11:         for each Particle do
12:             updateVelocity
13:             setNewPosition
14:             evaluatePopulation                  ▷ % Assign particle's fitness
15:             updatePbest
16:         end for
17:         updateGbest
18:     end while
19: end procedure
```

Velocity Update Equation

The i^{th} particle's updated velocity is calculated as follows:

$$v_i(t+1) = \omega.v_i(t) + \varphi_1.r_1(pbest_i(t) - x_i(t)) + \varphi_2.r_2(gbest(t) - x_i(t)) \ , \qquad (2.1)$$

where ω is a parameter specifying the particle's inertia, which determines how strongly the particle maintains its previous velocity (i.e. the higher the inertia the slower velocity changes). The real numbers r_1 and r_2 are chosen randomly within an interval (typically between 0 and 1), and φ_1 and φ_2 are the acceleration coefficients. The effects of these parameters are explained in detail latter in section 2.1.2.

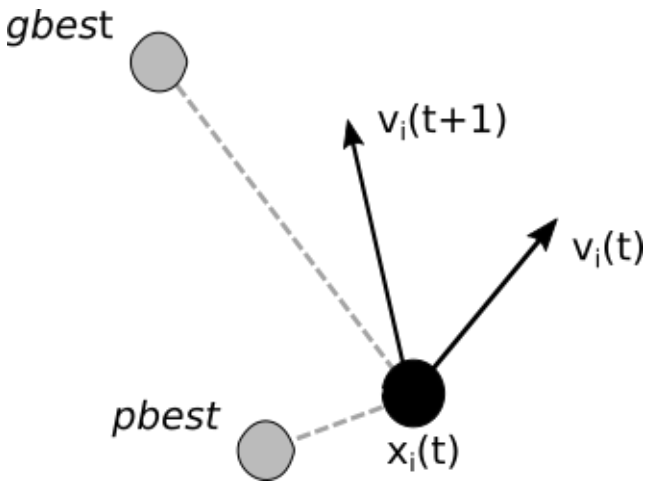

Figure 2.1: Graphical visualization of the velocity update of a particle (x_i).

Figure 2.1 shows a graphical visualization of how the particle's velocity is updated. In that figure, the new velocity $(v_i(t+1))$ is affected by its previous velocity and the *gbest* and *pbest* components.

It is also common to restrict the maximum velocity $(v_{max} \in [-v_{max}, v_{max}])$ to prevent instability. Note that the particle's maximum velocity may be static, or calculated dynamically [24].

New Position Equation

After updating its velocity (Equation 2.1), the particle's new position is calculated according to Equation 2.2.

$$x_i(t+1) = x_i(t) + v_i(t+1) \ . \tag{2.2}$$

This way, the particles drift through the space naturally, finding locally optimal points. Because they are attracted both to their own best position and the overall best position, over time a consensus may emerge as knowledge of the most promising point point in the search space spreads through all neighborhoods. This process will often result in convergence.

2.1.2 Parameter Selection

Inertia Weight (ω)

The inertia weight (ω) was not part of the Standard PSO algorithm when it was first introduced, but was added to the velocity equation in a later paper [25]. Its motivation is to balance exploration and exploitation in the search space. The inertia weight can be either a positive value, or a positive linear or non-linear function of time. Currently, the

most common choice is an inertia weight that linearly decreases from ω_{max} to ω_{min} as the simulation runs, as specified by Equation 2.3.

$$\omega = \omega_{max} - \frac{\omega_{max} - \omega_{min}}{maximumIterations}.currentIteration\ . \tag{2.3}$$

In practice, the inertia weight determines the influence of each particle's previous velocity upon its velocity in the next time-step. That is, the higher the inertia weight is, the higher will be the influence of the particle's previous velocity and, the smaller will be the step size.

Being a factor that is multiplied to the velocity update equation (Equation 2.1), ω is often a problem-independent parameter. Thus in practice performance often benefits from tuning the values of ω_{max} and ω_{min}.

Aceleration coefficients (φ_1 and φ_2)

The acceleration coefficients, φ_1 and φ_2, determine the intensity of the particles' attraction to *pbest* and *gbest*, respectively. The φ_1 acceleration coefficient is also called the cognitive acceleration coefficient because it is related with the particle's self awareness, i.e. it attracts it to its own best position found thus far. On the other hand, the φ_2 coefficient is known as the social acceleration coefficient, because it pulls the particle to the best position found by the whole swarm. Together φ_1, φ_2 and ω balance exploration and exploitation in standard PSO.

If φ_1 is much higher than φ_2 the particle will tend to wander narrowly around its best position, and will not explore areas of the search space far from that position. However, if the opposite happens (if φ_2 is much higher than φ_1) that can lead all particles to prematurely converge to the best value found so far, which often will not be the global optimum.

In most experiments the same value is used for both φ_1 and φ_2. However, in some experiments, where the search space has many local optima, it is common to take the same approach as is done for the inertia weight. That is, these values can also be changed linearly with increasing iterations [4]. r_1 and r_2 are two random values, commonly in the range [0, 1] that introduce some chaos.

2.1.3 Swarm Topology

Prior to Particle Swarm Optimization, many scientists developed a number of simulations about their interpretation of the movement of birds and fish. Reynolds's models [19] were motivated by his interest in the aesthetics of bird flocking choreography, while the goal of Heppener and Grenander simulations [20] was to discover the underlying rules that enabled large numbers of birds to flock synchronously. Hence, it is clear that, although the main goal of PSO was not to study biological events, being influenced by simulations

developed for the study of species social behavior makes the interaction between particles one of the core aspects of the algorithm.

It is very common to organize the particles in specific structures called neighborhood topologies. When a neighborhood topology is used, each particle belongs to a subset of the population and can only communicate with particles in the same subset i.e. its neighbors.

Although the terms neighborhood and neighbors suggest some kind of spatial proximity, most commonly such proximity is not guaranteed. In the most commonly used neighborhood topologies, the neighbors of each particle are defined and fixed in the beginning of the simulation. As the simulation proceeds and the particles' locations diverge, each particle's neighbors remain fixed, even if their locations are distant from one another in the search space.

The most common way to represent the neighborhood topologies is through a graph, where vertices represent the particles, and edges represent that the two linked particles are neighbors. As the relations between the neighbors are almost always symmetrical i.e. each particle offers its information to its neighbors and also receives from the same set of neighbors, the graphs are usually not directed. It is also important to notice that the neighborhood graph is connected. That is, every two particles are linked by a path of adjacent edges, causing each particle to be somewhat influenced by all other particles within the swarm.

Note that introducing a different neighborhood topology does not change the PSO algorithm or any of its equations. The main difference relies in the concept of *gbest*. When using a neighborhood topology, the particles do not have direct access to the information of swarm as a whole, only to the information relayed through its immediate neighbors. In such cases, in the velocity update equation (Equation 2.1) *gbest* is substituted by *lbest* (local best), meaning the best solution among a particle's neighbors.

The goal of introducing different neighborhood topologies is to shape how information flows in the swarm. Because each particle communicates only with its neighbors, it is intuitive that in denser topologies, where each particle has a larger number of neighbors, information will flow more quickly, and thus the knowledge of better positions will be shared more rapidly. The flow of information has direct influence on the performance of PSO.

Next, the most used neighborhood topologies and its influence on the swarm dynamics are explained in detail.

Neighborhood topologies

There are three main factors that affect the flow of information through the swarm [26]. The first factor is the degree of connectivity among particles in the swarm, i.e. the number of neighbors k. The second factor is the amount of clustering [26], meaning the propor-

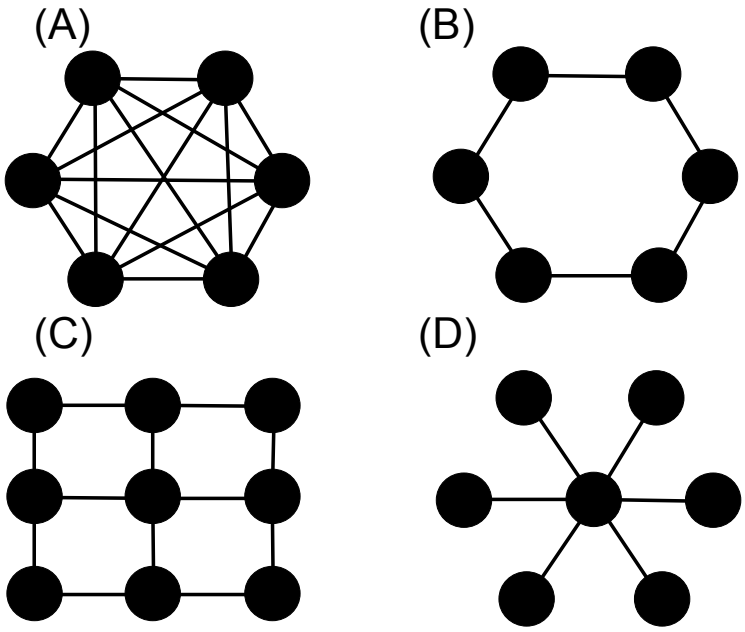

Figure 2.2: Graph examples of topologies used in PSO: (A) Fully-connected; (B) Ring; (C) Von Neumann; (D) Star.

tion of common neighbors shared common between each particle and its neighbors. The final factor is the average shortest distance from one node to the others in the topology's graph. Varying these parameters changes the way that particles interact with others and, consequently, the results of the optimization.

In the Fully-connected topology (Figure 2.2 A), also known as the *gbest* version of PSO, each particle is directly connected to all other particles in the swarm, making the neighborhood of each particle inclusive of the whole swarm. Thus, information about promising search points propagates immediately. Even though this topology is the most common one, its structure causes information to flow very quickly, which often results in fast convergence [21]. When optimizing multimodal functions, fast convergence can undermine successful optimization [27]. In such cases, topologies that slow the spread of information are often used.

Another common topology is the Ring topology, also known as the *lbest* version of PSO (Figure 2.2 B). In this topology, particles can communicate their best-encountered search points only with their immediate neighbors in a ring structure (i.e. each particle is arbitrarily assigned an order in a circular list and is connected only to the particles immediately in front and behind it in the list). As a result, information flows much more slowly than in the Fully-connected topology. In this way, one group of particles can converge around one point in the search space, while other groups can converge to other points or maintain divergence.

Another common topology is the Von Neumann topology (Figure 2.2 C). In this topology, each particle is assigned a location in a two-dimensional grid and is directly con-

nected to its four immediate neighbors. In other words, a visual representation of this topology is a grid in which each particle is connected to particles above, below, right and left of it. This topology can be seen as a compromise between the Fully-connected and the Ring topologies, because information flows faster than in the Ring, but not as fast as in the Fully-connected topology.

The last common topology is the Star topology (Figure 2.2D). It is a centralized topology, where all information passes through a single central individual. This way, all particles ”follow” the central particle, where the central particle serves as a buffer slightly slowing the speed of information transmission.

2.1.4 Premature Convergence in Particle Swarm Optimization

In itself, convergence can be a good feature in search algorithms. Particularly in PSO, this property enables particles to search more thoroughly in areas of the search space near high-fitness solutions, potentially allowing PSO to more efficiently reach an optimal solution. However, while PSO is often an effective optimization method, convergence can cause it to fail when dealing with problems with many local optima.

One of the key aspects in many search methods, including PSO, is managing the trade-off between exploration and exploitation. In the exploration phase the swarm will spread through the space while following the gradient of increasing fitness, which will enable uncovering promising areas. Meanwhile, in the exploitation phase, the search will tend to converge. All particles will approach the same high-fitness area, and search that particular area in more detail.

In cases where the problem has many local optima, following the gradients of increasing fitness can lead the search away from the global optimum. In these problems, particles may converge to the best solution found thus far. However, this may not be a desirable solution (i.e. not near the global optimum), causing the swarm to become trapped and unable to explore other areas of the search space.

2.1.5 Variations of Particle Swarm Optimization

The challenge of premature convergence in PSO is well-known [3, 4]. For that reason, since its inception in 1995, many researchers have presented variations to the standard algorithm. The changes range from minor parameters adjustments to complete overhauls of the PSO algorithm. Some examples of the proposed techniques involve, adding chaos [28], adaptive inertia and velocity weights [24], implementing different variables to measure the diversity and promote its increase or decrease at each point [29]. Most variations share the same intuitive motivation: Through actively maintaining a diverse population the search will converge more slowly, resulting in more thorough exploration of the search space.

Next, Barebones PSO, one of the most popular and effective variations to the algorithm will be presented.

Barebones PSO

In 2003, after noticing that the plot of a particle's position moving in one dimension appeared to be normally distributed, Kennedy proposed *Barebones PSO*, a variation of the standard PSO algorithm [6].

Kennedy's first motivation was to minimize the dependence of PSO's performance upon well-tuned settings of parameters such as ω, φ_1 and φ_2, which are otherwise required for high performance with the standard PSO algorithm. While other PSO variations have similar motivation, Barebones PSO is more radical and completely eliminates these parameters from the algorithm. In this variation, the particles move accordingly to a probability distribution instead of through the addition of velocity as before. Thus, the main difference from the standard PSO algorithm is the calculation of the velocity adjustment (Equation 2.4) where $\sigma = |pbest_{ij}(t) - gbest_j(t)|$ is the deviation, and the update of the new position (Equation 2.5).

$$v_{ij}(t+1) \sim N\left(\frac{pbest_{ij}(t) + gbest_j(t)}{2}, \sigma\right) , \tag{2.4}$$

$$x_{ij}(t+1) = v_{ij}(t+1) , \tag{2.5}$$

The velocity of each particle is thus sampled from the Gaussian distribution as shown above (Equation 2.4). Then this velocity no longer serves as a step size as in standard PSO algorithm, but as the new position (Equation 2.5). Barebones PSO favors exploration in the earlier stages of the simulation because initially, the *personal best* positions will be further from the *gbest* one, leading to higher deviations. In practice, particles will make bigger steps leading to more space being searched. As the simulation proceeds, the deviation will tend to approach zero, meaning that particles will make smaller steps, making the search focus on exploitation.

Even though it has considerably fewer parameters than standard PSO, making it simpler and easier to apply, it is still able to outperform the standard PSO algorithm in various types of problems [6, 30, 31].

2.2 Evolutionary Computation

Using computational models, Evolutionary Computation (EC) applies the principles of Darwinian evolution to solve problems. In 1859 Charles Darwin presented the modern theory of evolution [32]. In his book Darwin describes diversification in nature as a result

of natural selection, that through this biological process species evolve and over generations adapt to the environment.

Although this is not universally accepted [33] in EC the main principle taken from biology is that evolution optimizes fitness. That is, by natural selection, the most fit organisms will produce more offspring and over time, the less fit organisms will be displaced. The main concept in EC is that the idea of fitness in nature can be abstracted as a fitness function, and an evolutionary algorithm can be treated as another black-box optimization algorithm. Because typically they search for a particular objective when optimizing a fitness function, Evolutionary Computation simulations, called Evolutionary Algorithms (EAs) can be called objective-based algorithms.

EAs are a very popular, efficient and adaptive search mechanisms. They have been applied with success to practical problems such as numerical optimization [34], classification and machine learning [35] or automatic design [36].

In the following section a description of a generic EA is provided.

2.2.1 Evolutionary algorithms

EAs are population-based search algorithms that simulate the biological processes of selection, reproduction and variation to evolve a population. There are three primal variants of EAs: Evolution Strategies (ES) [37], Evolutionary Programming (EP) [38] and Genetic Algorithms (GA) [39]. Genetic Programming (GP) [40] has been developed more recently and because of its success in evolving computer programs to solve computational tasks, has been adopted as a major class of EA. A more detailed review of GP is provided in section 2.2.2.

Their strong inspiration from biology greatly influenced the terminology of EC. Given a particular problem, a candidate solution is called an *individual* (x_i), and a *population* (P) characterizes the entire set of current potential solutions. Each individual has a *fitness* value which is the result when it is applied to a fitness function ($f(x)$). This value serves as a grade indicating the quality of that candidate solution in relation to a specific problem. In EAs an individual is represented by a *genome* or *chromosome* which is the sequence of *genes* that describes it. Throughout the simulation the individual's genome changes producing a new candidate solution, this process is called *breeding*, and the new candidate solution is a *child* or *offspring*. A *new generation* is a population of offspring which replaces the previous population.

In the Algorithm 2 is a representation of a general EA.

In the initialization phase, the population of individuals is created. The number of individuals (the population size) is a parameter set by the experimenter and usually the population is initialized randomly. Then, the population is evaluated and each individual is assigned a fitness value.

Next, the most fit individuals (i.e. the individuals with the highest fitness values)

Algorithm 2 General Evolutionary Algorithm

```
1: procedure EA–PROCEDURE
2:     % Create and evaluate initial population:
3:     initialize
4:     evaluate                                          ▷ % assign fitness
5:     % Population's evolution:
6:     while CriteriaAreNotMet do
7:         selectParents                  ▷ % select individuals to be parents
8:         applyVariation                    ▷ % apply variation techniques
9:         createNewPopulation   ▷ % replace some/all individuals with the new
   generation
10:        evaluate                                      ▷ % assign fitness
11:    end while
12: end procedure
```

are selected and an offspring population is generated through variation operators. The most commonly used variation operators are sexual recombination (most often called crossover) and/or mutation. Finally, this new population is evaluated. The process continues until a sufficient solution is found or the allotted budget of evaluations is exhausted.

Selection

The selection mechanism in an EA is inspired by the process of natural selection in biology. The majority of evolutionary algorithms are objective-based, which in this case means that the objective-based fitness value of each individual is what determines if it will be selected or not. This means that individuals that perform ”better” at the task are more likely to continue in the population and produce more offspring. The idea is to remove less fit individuals from the population and only choose the successful ones to reproduce and propagate their genetic material to the new generations.

Besides *roulette wheel selection* [41] which is used later in the evolutionary experiments, many other selection techniques have been proposed, including *linear ranking* [42], *fitness-proportional* [39] and *tournament selection* [43].

In the *roulette selection* the probability (p) of an individual (i) to be selected in a population of N individuals is associated with its fitness (f_i). The relation between the probability and the fitness value of the individuals is expressed by the Equation 2.6.

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_j} \,. \tag{2.6}$$

The analogy to a roulette wheel is that each individual represents a section of the wheel proportional in size to its fitness. Thus the probability that individual will be selected increases as its fitness increase relative to other members of the population. This way, while individuals with higher fitness are more likely to be selected, less-fit individuals

still sometimes reproduce.

The most typical implementation of an Evolutionary Algorithm is to have each iteration correspond to a separate generation. At each iteration all (or at least most) of the population is replaced by a new generation of individuals. In respect to the selection technique this means that the fitter individuals are selected, according to the selection technique used, and then reproduce offspring. This generational approach is the one used later in the evolutionary experiments in chapter 4.4. Note that a *steady state* approach can also be used, where only some individuals are replaced at each time. More specifically, the selection technique picks the best individuals to produce a number of new individuals. Then, the same number of individuals are selected from the population, and replaced by the offspring. The individuals to be replaced can be the most unfit ones, but because this can compromise the population diversity, they are often selected at random. With this technique, each individual is evaluated only once in the whole simulation, but because the majority of the population is not replaced as in the typical EA implementation, the overall number of evaluations is smaller making it a faster and simpler approach.

Crossover

The sexual recombination or crossover is a variation technique that operates in multiple individuals (normally two). It performs by taking parts of each parent's solutions and combining them to produce a new offspring. Consequently, the new individual will contain some characteristics from each of its parents. Even though single-point crossover is the most common, other techniques of crossover can be used, including, two-point, uniform or flat crossover [44].

The crossover technique is very accepted and many researchers have proven the importance of this in evolutionary algorithms [45]. It is based on the idea that by exchanging information between fitter individuals it will produce an even better offspring. This way, the population will converge faster to the solution. However, if all members of the population are near the same area of the search space, if the algorithm can not ensure a certain degree of diversity in the population, premature convergence to a non-optimal solution can occur.

Mutation

In biology, an individual's genetic material can randomly change (mutate) due to many different reasons (e.g. mutations induced by chemicals or radiation, errors in the synthesis of DNA chains, etc). This mutations introduce some chaos to the evolution process.

As mention earlier, crossover ensures that the information of fitter individuals will be propagated throw-out generations. Consequently, these solutions will be increasingly similar among the population and some important characteristics may get loss. In EAs,

the mutation operator randomly alters some small proportion of the solutions, this way diversity between generations is maintained [46].

2.2.2 Genetic Programming

Genetic Programming (GP) was introduced in the early 90's by J.Koza [40]. Since then GP has become one of the most popular EAs. It evolves populations of computer programs stochastically by transforming them into new populations, choosing high-fitness individuals to reproduce, aiming to create increasingly fit populations.

In GP each individual is represented as a variable-sized parse tree instead of a fixed-length linear genome as in the algorithms described previously. The programs are expressed as these trees and, once compiled and executed, represent a potential solution to a specific problem. The fitness value is used to quantify how close is this solution from the ideal one. Then the fitter solutions are selected and the same reproduction operators as in EAs are employed (i.e. crossover and mutation) to produce the next generation. Although these operators have to be applied differently once they operate in trees instead of strings. Take the crossover, for example, this is applied to sub-trees. This means that whole sub-trees can be switched from two parent trees. Thus, the way that GP creates and evolves a population is by manipulating the parsed trees that represent individuals.

Figure 2.3 shows an example of the tree representation of the program **x - y + x**. In GP the leaves of the trees contain the *terminal symbols* which in this example are (x, y). The arithmetic operations (+, -), in the internal nodes, are called *functions*.

In the GP literature often a *prefix* notation is used in order to make it easier to see the relationship between the expression and the tree representation. In the example shown in Figure 2.3, the expression **x - y + x** becomes **- x (+ y x)** according to the prefix notation.

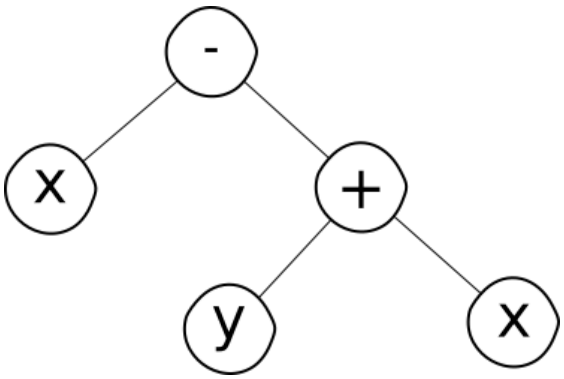

Figure 2.3: Example of the program **- x (+ y x)**.

The GP implementation depends a very much on the programming language that is being used. Programming languages that provide dynamic lists as a data type are easier to implement GP in. Others will require the researcher to implement lists or trees or use some already implemented library that provides them. Also, the tree representation can

be, in some cases, inefficient, once it requires management and storage of pointers in order to create an manage the trees. In this cases other EAs with linear representation may be preferred.

2.3 Grammatical Evolution

Grammatical Evolution (GE) is an Evolutionary Algorithm introduced in 1998 able to generate and evolve computer problems in an arbitrary language [47]. Since its invention, GE has been applied with success to many problems in various domains (e.g. financial and biology modeling [48, 49] and designing tools [50]).

Grammatical Evolution can be seen as a variant of Genetic Programming, but in addition to changing how an individual is represented, GE also differs in applying a genotype-to-phenotype mapping based on context-free grammars. Recall that GP uses parse trees to represent the programs (or solutions). Then the evolutionary process is performed on the actual programs. Thus, to meet their needs on a specific problem, many researchers end up developing their own programming language, usually a complex and time consuming process.

In GE, instead of a parse tree, the individuals are represented by a variable-length binary string genome. Each group of bits in the genome (commonly 8) represents an integer. These integer values, applied to a mapping function, dictate the selection of the appropriate rules from the grammar. By performing the evolutionary process in simple binary strings, GE is able to produce code in any language, when given an appropriate grammar.

In the following topic the Backus Naur Form grammar notation is described. Next an overview of the GE mapping process and its resemblance to the biological process of protein synthesis is reviewed. In order to clarify same aspects of the GE mapping process, a practical example is also provided.

2.3.1 Backus Naur Form

The Backus Naur Form (BFN) was introduced firstly by John Backus but it was not until Peter Naur's improvements that it became popular. BNF is a notation capable to express grammars without context needed. It decomposes the grammar in fragments often called *derivation rules* or *production rules*. These allow the composition of a syntactically correct program in a given language.

A BNF grammar is represented by:

- Sets of *non-terminal* (NT) symbols typically represented inside angle brackets ($'' <>''$);
- Sets of *terminal* (T) symbols;

- A *start* symbol (S) which is a member of NT;
- The *production rules* (P) which are used to map the non-terminal symbols into terminal symbols.

A non-terminal symbol can be mapped into a non-terminal symbol, a terminal symbol or a composed expression of terminals and/or non-terminals symbols. It can also have various alternatives and be mapped in different symbols according to the decoder value. In this case, the different alternatives are separated by vertical bars $''|''$.

2.3.2 GE Algorithm and Mapping Process

GE is set up in two independent parts: the *search mechanism* and the *mapping process*. In the first an Evolutionary Algorithm or other population-based algorithm is used as a search mechanism, i.e. it creates and evolves the population. In the (genotype-to-phenotype) mapping process the individual's binary string (genome) is used to determine which production rules from a BNF grammar definition are used to generate the program.

The GE mapping process is inspired by the biological process of protein synthesis from an organism's genetic material (DNA). Ultimately, these are responsible for the manifestation of some traits (e.g. fur or petals color, size of a stem [51], etc), and in conjunction with environmental factors constitute the phenotype. In GE the phenotype is the resulting program.

Figure 2.4 compares the GE and the biological organism's mapping process. The individual's genotype (binary string) in GE corresponds to the living organism's DNA (double helix). Through *transcription* the genotype becomes a integer string mimicking the RNA (single helix) in biology. This step is not mandatory in GE since machines easily compute binary digits, it is mostly done because integers are more intuitive to humans to understand. In GE, the program (phenotype) results via application of production rules and, in the biological case, the protein results of the order and type of aminoacids that are joined together.

The following steps take place before evaluating each individual and correspond to the GE mapping process.

1. The algorithm reads the codons of typically 8 bits and transforms them into integers;
2. Each integer corresponding to the codon bit sequence determines which production rule from the BNF grammar is chosen according to: $Alternative = CodonValue\%Num.of Alternatives$;
3. When the genotype has fewer codons than non-terminal symbols, the genome is *wrapped*, i.e. the algorithm continues reading again from the beginning of the genotype;

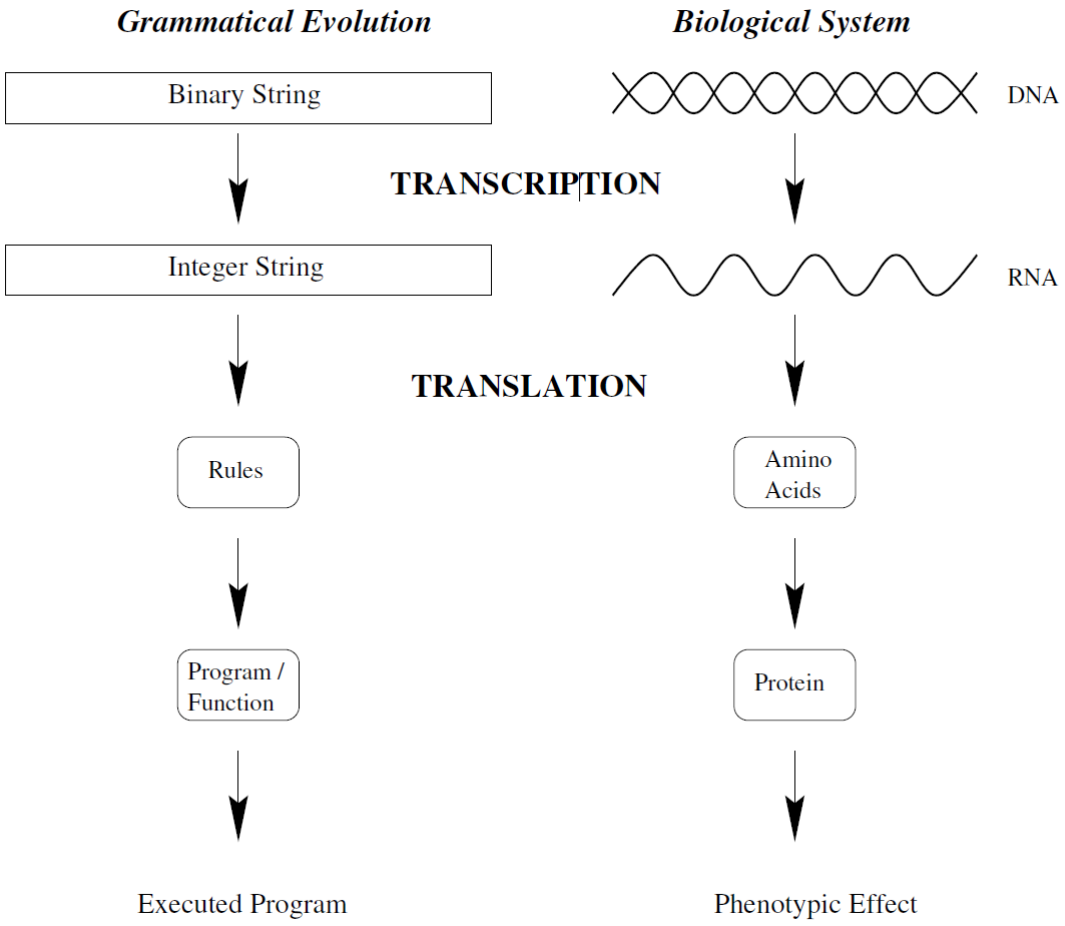

Figure 2.4: Comparison of the GE mapping process with the biological protein synthesis process. Figure from [16].

4. The last points repeat each if non-terminal symbols exist;

Next, a concrete example of a mapping process is provided, demonstrating in detail how a program is generated.

Mapping Example

In the GE mapping process (i.e. the process by which it decodes a genome into a program), an individual's genotype is used to construct a program (which can then be evaluated in the domain).

An example of the mapping process in GE is shown in Fig. 2.5. The BNF grammar showed in that figure represents a set of rules to construct a simple mathematical expression. This grammar is composed by a set of three *non-terminal* symbols: $NT = \{< O >, < E >, < V >\}$; a set of four *terminal* symbols: $T = \{+, -, x, y\}$ and finally, a *start* symbol: $S = \{< E >\}$. It is trivial that any program resulting from this grammar

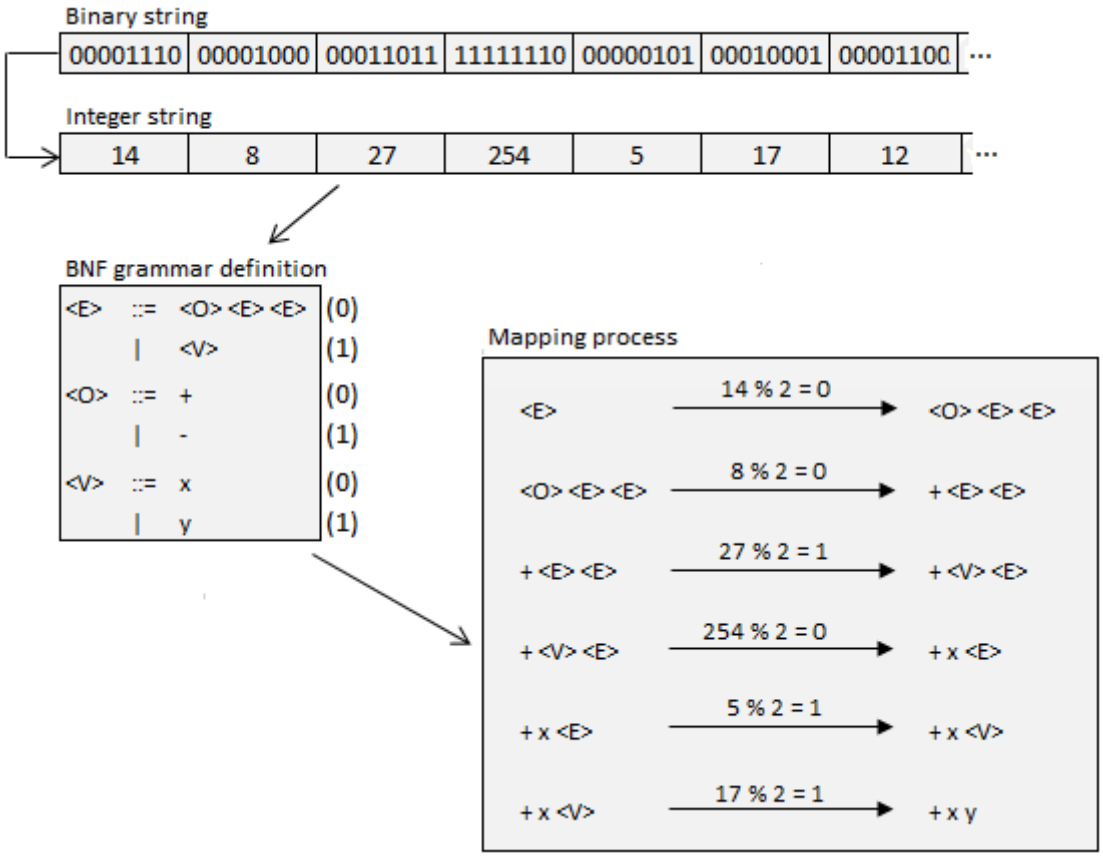

Figure 2.5: GE and GS mapping process. This example is based on [16].

will only have the symbols contained in it. This grammar have three production rules and, in this case, all of them have two alternatives that can be selected.

The algorithm reads the chromosome sequentially from left-to-right and always chooses the left-most non-terminal symbol to be derived. In this example, the first codon - 00001110 - is converted to its decimal value - 14 (integer representation is used for the remainder of this example for ease of understanding). From the underlying grammar, the start symbol <E> has two possible alternatives that can be applied. The mapping function in Equation 2.7 is used to determine which alternative is selected. Thus, the modulus of the value of the codon with the number of alternatives determines which alternative is selected.

$$Alternative = CodonValue\%Num.of Alternatives. \tag{2.7}$$

For this example, **14** $\%$ **2** equals 0, meaning that the rule <O> <E> <E> is selected. The expansion of this rule results in three non-terminal symbols. Note that the mapping process continues until only terminal symbols remain. Next, the leftmost symbol - <O> - and the second codon - 00001000 - which represents the decimal value - 8 -are processed in the same way. Meaning **8** $\%$ **2 = 0** selecting the symbol "+" which is a terminal. At this

point the algorithm returns to the next left-most non-terminal and the nest codon, in this case the <E> symbol and the codon 27. Because **27%2 = 1**, the selected alternative is the <V> non-terminal symbol. The next codon - 254 - will select the alternative to decode <V> which is "x" (**254 % 2 = 0**). Recall that, it still remains a non-terminal symbol to be decoded - <E>. Using the next codon - 5 - the symbol <V> will be selected (**5 % 2 = 1**). Finally, **17 % 2 = 1** meaning that the last symbol is "y". The resulting expression, which results from applying the grammar in Fig. 2.5 to a individual with the genotype present in the same figure is: **" + x y "**.

In this example the genotype has enough codons to decode all non-terminals, but that is not always the case. When the genotype has fewer codons than non-terminals, the genome is "wrapped", i.e. the algorithm continues reading again from the beginning of the genotype. The number of times the genotype wraps is limited in practice to mitigate infinite cycles. In this case, sometimes non-terminal symbols may remain without elements of the genotype to resolve them and the respective individual is marked as invalid.

2.4 Grammatical Swarm

Grammatical Swarm (GS) is an Evolutionary Algorithm introduced by O'Neill and Brabazon in 2004 [16]. Because it is a biologically-inspired algorithm that involves swarms, GS is also a form of social learning, or, more commonly called *Swarm Programming*. The Grammatical Swarm algorithm combines the Particle Swarm Optimization search algorithm (reviewed in section 2.1.1) with the genotype-to-phenotype mapping of Grammatical Evolution (section 2.3.2).

When reviewing Grammatical Evolution in section 2.3, it was mentioned that its algorithm can be divided in to two different and independent parts: the *search mechanism* and the *mapping process*. The GS implementation is similar to the one in GE as it also uses a search mechanism and a mapping mechanism to produce program solutions. The most common GE applications use a Genetic Algorithm as the search mechanism. That way, the population of individuals is evolved based on the metaphor of natural selection, and the reproduction operators such as mutation and crossover are used in order to produce fitter individuals. Then, a BNF grammar is used to develop compilable programs which are, ultimately, potential solutions to the problem.

In GS, instead of a GA the PSO algorithm serves as the search mechanism to create and evolve the population. The main difference is that no reproduction operators are used. In this case, instead of a population of individuals (as in an EA), the population is composed of particles, and instead of a binary string representation, each particle is associated with two d-dimensional vectors, one recording its position and the other its velocity, as in PSO. The real-valued elements of each particle position must be rounded and only then we have a vector composed of integer codons that can be transcribed into a

program using a grammar.

Even though it is a recent method, GS shows significant potential [16]. Its fixed-length vector representations and few parameters make GS faster and easier to implement and understand than other Evolutionary Algorithms, while providing, in some cases, equivalent or better results.

2.4.1 Additional constraints

Even though coupling the PSO algorithm with the GE mapping process is trivial, in GS some parameters have added constraints which are reviewed in the following topics.

Fixed Length Particles

In a tree-based GP, the length of an individual's genotype is not fixed, meaning that it can adjust along the simulation, by increasing or decreasing in size, between the maximum and minimum values set. In GS, the size of the position and velocity vectors (usually called *problem-dimensionality*), stays fixed during the simulation.

Real Valued Position Vector

Note that the PSO equation of the velocity update (Equation 2.1) have parameters that can have floating point numbers, such as r_1 and r_2. This means that because the new position is calculating by adding the velocity update to the current position (Equation 2.2) the particle's position vector is always composed by real-valued numbers. Because the GE mapping process only accepts integer codons, this means that values on each dimension of the position vector have to be rounded, up or down, into the nearest integers. This way the mapping process can remain the same as it is in GE, i.e. after rounding, the particle's position corresponds to a program, much like the genotype in GE.

Position and Velocity Restriction

The position vector is restricted so that each dimension only contains values in the range $[C_{min}, C_{max}]$. As shown in Algorithm 3, when the new particle's position is calculated (Equation 2.2) if any dimension is lower than C_{min} or higher than C_{max}, it assumes the values of C_{min} and C_{max}, respectively.

Algorithm 3 Domain coherence

```
1: if [dimension i] of position-char < (C_min) then
2:     set [dimension i] of position-char C_min
3: end if
4: if [dimension i] of position-char > (C_max) then
5:     set [dimension i] of position-char C_max
6: end if
```

A maximum velocity is imposed so that the maximum distance a particle can move in a single step is controlled. This way, each particle can only make steps smaller than v_{max} in any direction: $(v_{max} \in [-v_{max}, v_{max}])$. The motivation is that excessively large step sizes can destabilize the algorithm.

2.5 Novelty Search

The Novelty Search (NS) approach was introduced by Lehman and Stanley in 2008 as an alternative to objective-based optimization in Evolutionary Computation [7]. The main goal was to overcome deception and create a practical open-ended evolutionary algorithm.

The key idea behind novelty search is that it ignores the objective of the search, and instead rewards individuals that demonstrate novelty relative to individuals previously encountered in the search. Although ignoring the desired objective may seem counter-productive, in deceptive domains novelty search has often outperformed objective-driven algorithms [52, 53]. The insight is that by not optimizing a measure of progress towards the objective, novelty search is not susceptible to premature convergence to local optima. Of course, the significant trade-off is that the search as a whole becomes less explicitly controlled.

In the following section the background of Novelty Search is reviewed illustrating the struggle of objective-based search to overcome deception. Then the principles of the NS approach are described in detail, and finally the novelty search algorithm is presented.

2.5.1 Deception

Many researchers in EC are interested in why Evolutionary Algorithms fail on certain problems, and how can that failure be avoided. One such branch of research studies of deceptive fitness landscapes. There is no overall consensus on the definition of a *deceptive problem*, which various researchers have described in different ways [54, 55, 56]. A simple way to define a deceptive problem is one in which following the gradient of the objective function does not lead to the ultimate goal but toa local optimum [9].

Many approaches to mitigate deception and prevent premature convergence have been proposed over the years. One class of approaches are diversity maintenance techniques [57, 58, 59], which are inspired by actively encouraging a diverse population. Many of these techniques operate by dividing individuals into subsets, often called niches (*niching*), wherein each individual only competes with others in the same niche. The *hierarchical fair competition* technique [57] for example, implements competition between individuals with similar fitness scores. Likewise, in the *age-layered population structure* approach [58] individuals with different genetic ages compete with each other. Moreover, *fitness sharing* [59] promotes competition among similar solutions resulting in a constant

pressure to find different ones. The *Fitness Uniform Selection Scheme* [60] is an interesting method that rewards individuals with different fitness values, ignoring the pressure to increase the fitness. In a way, this method resembles novelty search but instead of rewarding novel behaviors, it rewards novel fitness. Even though they apply different techniques, these methods have the same underlining idea that by increasing the exploration, promising areas of the search space will be found. The idea is to then exploit these areas to find a desirable solution. But because they rely on the objective fitness function, if local optima are pervasive or if differences in genotype do not correlate with behavioral differences, these methods may still be deceieved.

Another class of techniques, known as Estimation of Distribution Algorithms (EDA) [61, 55], try to model the fitness function to smooth the landscape in order to make it less deceptive. Still, if the objective fitness function is sufficiently uninformative, such modelling fails to reveal a successful individual.

Yet another popular approach is to successively apply different objective functions specifically crafted to avoid local optima [62, 63]. This involves a prior study that conflicts with the vision of learning to act without being programmed. Also, some problems have ambiguous objectives that can be very difficult or impossible to identify prior to the simulation.

2.5.2 The Novelty Search Approach

Novelty search is a relatively new approach to overcome deception. It differs from the previous approaches because it takes the radical step of ignoring the final objective, instead it searches only for instances with significantly different behaviors than those found earlier in search. For that, instead of the traditional objective function, NS uses a *novelty metric* which measures the novelty of an individual in comparison to other individuals in the search, according to its behavior. This enables rewarding individuals with novel behaviors, which can lead then to an exploration of the space of behaviors. This exploration can often lead to discovering the desired behavior as a side effect.

Even though NS explores the whole behavioral search space without any final objective rather than finding novelty, this approach is far from being similar to exhaustive search. In most practical domains the number of possible behaviors is not exaggerated and it is also limited. Recall that NS optimizes towards novel behaviors, these are related with a specific task that, on its own, usually provides enough constraints to the number of possible significant behaviors.

To better understand the difference between NS and exhaustive search consider the following problem: *Make a three-dimensional biped robot to walk as far as it can in a specific amount of time*. In this case, the only significant behaviors are the ones regarding the robot's locomotion. These are limited by the morphology of the robot and by physics. Thus, while the search space may be infinite (i.e. if there is no limit to the number of

codons in a individual's genotype, the EA can add new codons indefinitely and the number of codon combinations are infinite), the behavior space is limited. Suppose that the robot's behavior is characterized by its ending location after exhausting the time limit. In this case, the robot may arrive to a specific point many times through different paths, but because the final position is the same, all will be characterized as the same behavior. Ultimately, in this case, the number of possible behaviors is the number of different points where the robot may end up when the simulation ends.

Of course, without any pressure to optimize towards the objective, intuitively it seems unlikely that a raw search for novelty will be an effective algorithm for solving problems. Yet if measures of novelty are constructed in ways that capture important dimensions of behavior in the domain, the surprising result is a practical algorithm for solving deceptive problems [7, 9, 15].

2.5.3 Novelty Search Algorithm

Novelty search was proposed with Evolutionary Algorithms in mind. Changing a EA to become *novelty-driven* instead of *objective-driven* does not require many changes besides replacing the fitness function with a novelty metric.

Novelty search requires that each individual be assigned a behavior descriptor, which is a vector summarizing the individual's behavior. Thus to apply novelty search to a new domain requires the experimenter to devise a characterization of behavior in that domain. For example, in the biped robot simulation, mentioned above (section 2.5.2), the individual's behavior descriptor can be its [x,y] coordinates after exhausting the time limit. Because it biases the search, different behavior descriptors produce different results. Thus effective application of novelty search may depend upon a characterization of behavior that succinctly captures important aspects of behavior in a domain.

Problems with large behavior spaces can benefit from maintaining an archive of past behaviors. The idea is to prevent search from repeatedly cycling through a series of behaviors, reflecting a fleeting sense of novelty. By archiving past behaviors, novelty can be measured relative to where search has been and where it currently is, thus guiding towards behaviors that are genuinely novel. Two common archiving strategies are to add behaviors with novelty that is higher than a threshold value, or to select individuals randomly to archive.

To calculate the novelty of an individual it is necessary to first define the distance between its behavior descriptor and that of others in the population and in the archive. This distance is calculated using, most commonly, the euclidean distance, but the hamming distance or Fourier coefficients are also used [64].

A novelty metric measures how different is an individual's behavior from the others. Given the behavior descriptor and the distance metric between such descriptors, the novelty score is calculated as shown in Equation 2.8, where μ_i is the i^{th} nearest neighbor of

x.

$$\rho(x) = \frac{1}{k} \sum_{i=0}^{k} dist(x, \mu_i) \ . \tag{2.8}$$

The novelty of each individual is the average distance of its behavior to its k-nearest-neighbors. This way, individuals in a less dense area of the behavior space are given higher novelty scores, thus creating pressure in the search towards further novelty.

Chapter 3

Novelty-driven Particle Swarm Optimization

This chapter presents a new method for combating the challenge of premature convergence in Particle Swarm Optimization, by coupling it with Novelty search. The next sections introduce this method and formalize its algorithm. Its performance is then compared with control algorithms in three benchmark domains: the Mastermind Problem, the Santa Fe Ant Trail and the Maze Navigation Problem.

3.1 Introduction

The novelty search approach has been successfully applied in many fields of Artificial Intelligence and to many problems. In maze navigation, the Santa Fe Ant Trail and the Los Altos Hills Ant Trail, Lehman and Stanley [9] demonstrated that novelty search applied to Genetic Programming outperforms objective-based fitness by solving problems more frequently given the same budget of evaluations. Furthermore, novelty search is able to avoid the problem of bloat, evolving smaller programs than objective-based search.

Also in the field of GP, using the Santa Fe Ant Trail, Doucette and Heywood [65] studied the performance of novelty search and its capability to generalize. In their study, even though novelty search was less successful than objective-driven search, novelty search showed that it can evolve programs that generalize better. More recently, in 2013, novelty search with GP was applied to difficult classification problems and provided better results than the original objective-driven GP algorithm [66].

The first application of novelty search using Grammatical Evolution, by Urbano and Georgiou [15], compared the success rate and the quality of solutions in the Santa Fe Ant Trail. Novelty search significantly outperformed objective-driven GE in both these measures. Building upon the success of that work, an analogous method that drives PSO through novelty instead of the search for the final objective is proposed here, called Novelty-driven Particle Swarm Optimization (Novelty-driven PSO or NdPSO).

The key idea in NdPSO is that instead of particles being drawn towards high-fitness areas, they are drawn towards positions that represent novel behaviors. That is, the Equation 2.1 remains unchanged but the concept of "best position" changes. This way, in Novelty-driven PSO, the *pbest* position is not the position where the particle obtained best fitness but the position where it was most novel. Similarly, the overall best position (*gbest*) is the position where the maximum value of novelty was obtained. For this reason, these quantities are referred to as *pnovel* and *gnovel*, respectively.

Even though novelty is being maximized similarly to how objective-based fitness was, NdPSO does not simply replace one objective function with another. Objective-driven search is conceptually different from novelty-driven search. The first creates a gradient towards the objective with the purpose of bringing the search towards it. The second creates a gradient of behavioral difference that diverges from the past towards new discoveries. In this way, novelty is maximized without any concept of where the search should terminate or even what general direction it should take within the search space.

Thus the novelty of each particle, as well as the novelty of its *pnovel* and the novelty of *gnovel* will decrease as other particles approach similar behaviors, lowering the threshold for new behaviors to displace them. In addition, as in the novelty search algorithm described previously, often-encountered behaviors of particles are accumulated in a global archive that also is taken into account when calculating novelty. The overall effect is that particles are repelled from areas of the behavior space where search currently is and has previously been, thus encouraging particles to continually discover genuinely novel behaviors.

3.1.1 The Novelty-driven PSO Algorithm

Algorithm 4, shows the pseudo-code for novelty-driven PSO algorithm. Because merging novelty search with PSO does not require much change beyond replacing the objective-based fitness function with a novelty metric, the core of the algorithm does not significantly differ from the standard PSO algorithm.

One important aspect of novelty is that it is both relative and dynamic. That is, each particle calculates its novelty with respect to other particles' current behaviors and past archived behaviors. In this way, a highly novel behavior may become less novel over time as other particles become drawn to it and it is added to the archive. For this reason, the tracking of the *pnovel* and *gnovel* positions should take this dynamism into account. In particular, the novelty of each particle's *pnovel* and *gnovel* behaviors must be recalculated each iteration. At each time-step, after calculating the novelty of each particle, the novelty of the particle's *pnovel* behavior is recalculated using the current population and the archive. Then, the particle's *pnovel* behavior is updated, corresponding to the *updatePnovel* procedure in the Algorithm 4. In this step the *pnovel* changes if the particle's current behavior corresponds to a higher novelty than the current *pnovel* after being

Algorithm 4 Novelty-driven PSO algorithm

```
1: procedure NSPSOPROCEDURE
2: createPopulation: foreach Particle
3:     setRandomPosition
4:     setRandomVelocity
5:     evaluatePopulation
6:     setPbestCurrent
7:     executeNovelty
8:     setPnovelCurrent
9:     end foreach
10:    updateGbest
11: loop:
12:    if NotDone then foreach Particle
13:        calculateVelocity
14:        calculateNewPosition
15:        evaluatePopulation
16:        updatePbest
17:        executeNovelty
18:        updatePnovel
19:        end foreach
20:        updateGbest
21:        updateGnovel
22:    end if
23: end procedure
```

updated (Algorithm 5).

Algorithm 5 updatePnovel procedure

```
1: procedure UPDATEPNOVELPROCEDURE
2:     askParticle(i) [
3:         calculateNovelty of pnovel
4:         if (novelty-score of pnovel) < (novelty-score) then
5:             set pnovel Particle-behavior
6:         enf if
7:     ]
8: end procedure
```

After, the *gnovel* is also updated so that it will correspond always to the *pnovel* with highest score (Algorithm 6).

Algorithm 6 updateGnovel procedure

```
1: procedure UPDATEGNOVELPROCEDURE
2:     set gnovel behavior of (max pnovel)
3: end procedure
```

Such recalculation is not computationally expensive because no additional domain evaluations are required. Instead, the behavior vectors of each *pnovel*, *gnovel*, and their positions are cached, and the nearest-neighbor calculation is performed on those cached vectors as described above.

3.2 Proof of Concept

In this section NdPSO is tested in three deceptive benchmark domains: the Mastermind Problem, the Santa Fe Ant Trail and the Maze Navigation Problem. Recall that novelty search pursues novel behaviors, which makes it best-suited for deceptive problems in which it is possible and intuitive to characterize an individual's behavior. For this reason novelty-driven algorithms are unlikely to perform well in black-box optimization problems, because there is little information to base novelty upon beyond the output of the function itself. Thus the choice of test domains reflects the type of problem for which the approach is likely to be appropriate.

The aim of these experiments is to compare the performance of the objective-driven PSO method with the performance of the NdPSO algorithm proposed in this thesis. In the topics below (3.2.2, 3.2.3 and 3.2.4), the probability of finding an optimal solution (i.e. probability of success) during the simulations is compared between NdPSO, Standard PSO, Barebones PSO and random search, in three different domains. In those experiments a GP like encoding is being used because the programs are generated using the GP mapping process. Because bloat is a harmful dynamic in GP wherein the average size of the programs in a population grows rapidly without increasing performance, and this can potentially occur in NdPSO, a *program bloat* analysis is also presented in the same domains as before. This phenomenon is problematic because larger programs are more computationally expensive to evaluate, and also they are harder to interpret and may generalize poorly. In their experiments, Lehman and Stanley [9], suggest that novelty search is less susceptible to program bloat than objective-based search.

Based on previous success of NS applied to GE, in these experiments NdPSO was implemented as an extension of the Grammatical Swarm (GS), which is a grammatical implementation of PSO (Section 2.4), thus, this implementation is called novelty-driven Grammatical Swarm (NdGS). Therefore, this implementation takes advantage from the grammatical approach (see section 2.3), This way, instead of searching in a space of positions as traditional PSO, NdGS searches in a space of programs.

All experiments have been performed in Netlogo [67], an open-source agent-based simulator. Netlogo provides a programmable modeling environment for simulating natural and social phenomena, making it well suited for modeling complex systems that evolve over time. A Java implementation of the GE algorithm [68] was used - the jGE library - and the respective Netlogo extension [69] - jGE.

3.2.1 Experimental Settings

A succinct description of important parameters is presented next.

Maximum-iterations Represents the stopping criteria for the simulation, and the number of times that each particle is evaluated. If a particle reaches the goal before finishing the maximum iterations the simulation stops. In these experiments the particles are evaluated one additional time when they are created in order to have assigned fitness and novelty when the actual simulation starts.

Max-steps Used in the Santa Fe Ant Trail and in the Maze Navigation Navigation Problem, this parameter represents the maximum number of time-steps that an agent can make in the benchmark domain in each run.

Sample-frequency When an agent is running a solution in a benchmark domain, its behavior keeps changing. Take the biped robot example provided in section 2.5.2, if we consider as a behavior the robots's [x,y] coordinates, this means that every time the robot changes its coordinates, its behavior changes. The Sample-frequency is the frequency in which the current-behavior of the agent is sampled. That is, in the same example, if the sample-frequency is the same as the max-steps, the final behavior will be the robot's final coordinates. But if the sample-frequency is 15 and if the simulation has a max-steps of 615, this means that every 15 steps the robot's coordinates will be recorded in its behavior. In this case, its final behavior will be composed by 41 pairs of [x,y] coordinates.

As the previous parameter, this is only used in the Santa Fe Ant Trail and in the Maze Navigation Navigation Problem.

Dimensionality Is equivalent to the number of codons in a particle's genotype in a GA. In PSO, this value depends on the optimization problem, for example, considering the following function to optimize: **2x + 3y + z = 0**; in this case the problem-dimensionality is 3, which correspond to the number of variables to calculate in order to optimize the function. In GS and the NdGS implementation, the dimentionality corresponds to the number of codons of the particle's position vector.

Particle-inertia In these experiments the equation 2.3 in Chapter 2 is used to calculate the inertia. For the values of ω_{min} and ω_{max} it was used 0.4 and 0.9, respectively.

Always Valid As suggested by O'Neill in [16] we also experimented just considering positions that translate in valid programs. In this case, all individuals in the initial population are forced to be a valid program. This is achieved by generating random initial positions and evaluating them, if a position corresponds to an invalid individual, it is discarded and another position is generated.

During the run, if a particle updates its position and it results in an invalid program being produced, it recalculates its velocity update using equation 2.1 as normal, then it updates its position again. This means that each particle updates its velocity and position until a valid program is generated.

In the novelty-driven case to an invalid particle it is assigned a "dummy" behavior descriptor so that other can calculate the behavioral distance in the regular way.

Neighborhood Topologies Three different neighborhood topologies are tested: Fully-connected, Ring and Von Neumann.

As in O'Neill's original GS experiments [16], some parameters were adopted in all experiments. Maintaining these parameters allow us to fairly compare the method's performance. Table 3.1 presents the values of all fixed parameters used throughout these experiments. These were adopted according to [16].

Table 3.1: Fixed parameters used throughout the comparison of PSO and NdPSO

Parameter	Value
Initial population	random
Population-size	30
Dimensionality	100
Max-iterations	1000
Codon-size	8
Max-wraps	10
Maximum velocity	255
Attraction to *pbest / pnovel*	1
Attraction to *gbest / gnovel*	1
Inertia weight	dynamic
Minimum inertia	0.4
Maximum inertia	0.9

Random Search was implemented in a way that on each step, the particle's new position was randomly calculated using the random function in NetLogo as shown in 7.

Algorithm 7 updateRandom Procedure

```
1: procedure UPDATERANDOMPROCEDURE
2:     set i 0
3:     while i < dimentionality do
4:         set particlePosition i random(0 C_max)
5:         set i i+1
6:     end while
7: end procedure
```

3.2.2 Mastermind

In its original conception, mastermind is a code-breaking game where one player (the codemaker) chooses a configuration of four colored pins, while the other (the codebreaker) attempts to deduce the configuration given a limited number of attempts. The computational version considered here is similar, i.e. the task is finished if the correct sequence is found or the maximum number of attempts exhausted. As in O'Neill's formulation [16], four possible colors are considered, represented by the numbers 0, 1, 2 and 3. The correct code has eight pins and the solution was fixed to **3 2 1 3 1 3 2 0**. NdGS and GS both use the same grammar as in the original GS experiment - Fig 3.1.

```
<pin> ::= <pin> <pin> | 0 | 1 | 2 | 3
```

Figure 3.1: BNF-Koza grammar definition for the Mastermind problem.

The objective-based search requires an objective-based fitness function. In this domain, fitness is scored by the following function:

- One point is awarded for each correct colored pin regardless of its position.
- The number of points awarded is limited by the number of pins in the target sequence with that color.
- If all pins are in the correct position, an additional point is awarded.
- The fitness score is normalized by dividing the score by the maximum score possible (Equation 3.1). Note that *maxScore* in this experiment is 9.

If the particle corresponds to an invalid program, the fitness value is -1. If the resulting sequence has more than eight pins, it is cut and the fitness is calculated the same as before. By design, this problem is very deceptive, with strong local optima corresponding to all sets of correct colors in wrong positions.

$$fitness = \frac{score}{maxScore} \ . \tag{3.1}$$

In contrast to objective-based search, novelty search requires a characterization of behavior. In this problem two distinct characterizations were considered:

- **behaviorMm1** - the fitness value.
- **behaviorMm2** - a tuple of two integers consisting of the number of correct colors and the number of correct positions.

If the particle generates an invalid program, its behavior descriptor is [0] if behaviorMm1 is being used and [0 0] if behaviorMm2.

Two things are important to note. First, searching for novel values of objective fitness is different than simple objective-based search. That is, novelty search will be driven to accumulate all possible fitness values, not only the most promising ones. So the performance of objective-driven GS and novelty-driven GS may diverge even though they are given the same underlying behavior (i.e. in the objective-based case the search will try to follow space points that maximize the fitness function, where in the novelty-driven case, it will search for positions that translate in novel behaviors). Second, behaviorMm2 provides an ideal decomposition of the domain which is obscured by the objective fitness function (i.e. both colors and placements are important). Thus this characterization highlights the potential for injecting experimenter knowledge into the search process, which is otherwise complicated in objective-driven search due to the need to reduce performance information to a single number.

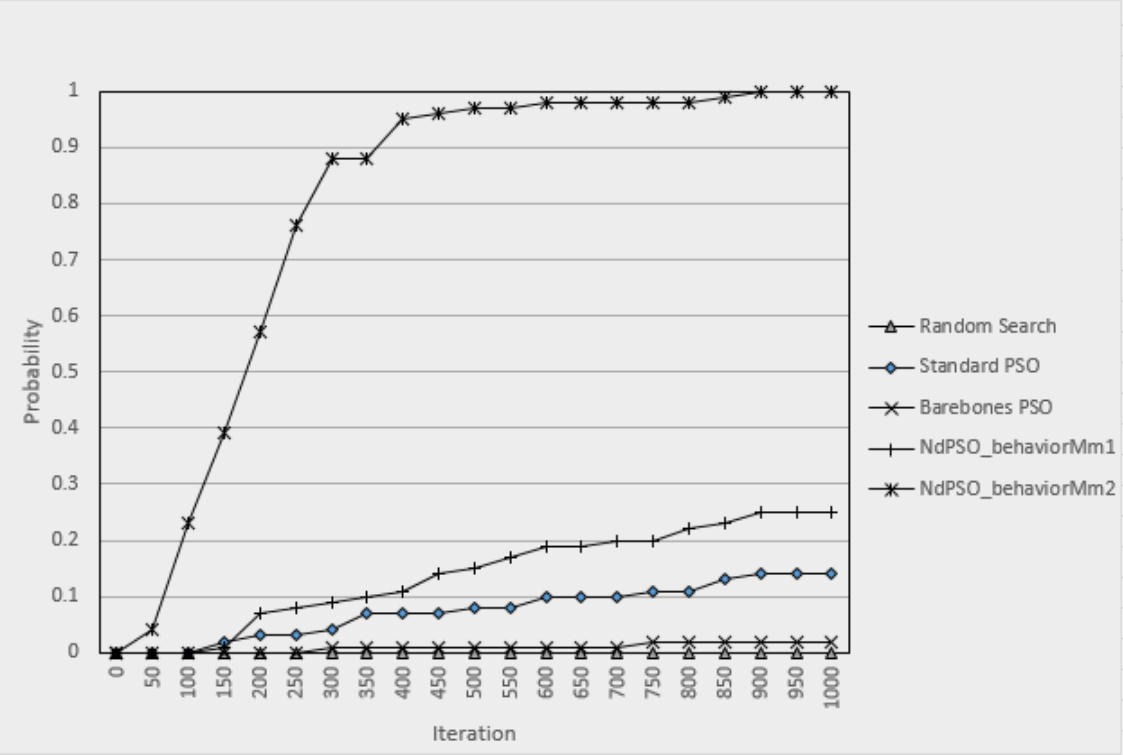

Figure 3.2: The probability for objective-based PSO, Barebones and novelty-driven PSO and random-search to discover solutions in the Mastermind problem.

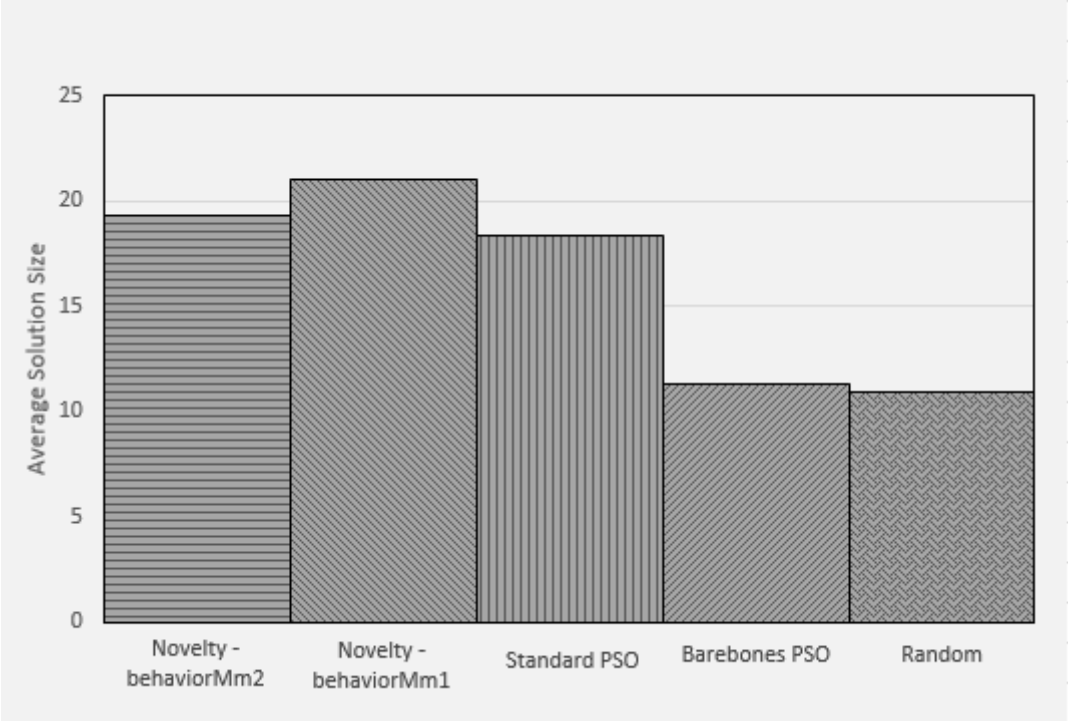

Figure 3.3: Bloat comparison in the Mastermind domain. Comparison of the average program size (number of characters without spaces) of the best solutions obtained.

Figure 3.2 shows a plot of the probability of each algorithm to find the solution in the Mastermind problem within 1000 runs. As can be seen, the final performance of objective-driven Barebones and random-search do not differ significantly; regarding NdPSO, even though behaviorMm2 performs much better than behaviorMm1, they both outperform all the objective-driven algorithms tested and the random-search algorithm.

A comparison of the average size of the best programs obtained with the different algorithms tested is presented in graph form in Figure 3.3. In this domain, the behaviorMm1 of NdPSO produces significantly larger programs than the objective-driven tested and random-search using a significance level of 0.05 in a t-Student test. Even though, behaviorMm2 of NdPSO produces significantly larger programs than the Barebones PSO and random-search using the same significance level and statistical test, the difference between it and Standard PSO is not significant.

3.2.3 Santa Fe Ant Trail

The Santa Fe trail is a difficult and popular benchmark in both GP [70] and GE [47]. It is considered a standard benchmark in both fields mainly because it is a very deceptive problem. The goal is to evolve a computer program that can efficiently guide an artificial ant to eat all pieces of food placed in the trail. Figure 3.4 shows its graphical representation. In this figure, the colored cells represent the actual trail, where the lighter ones are the food and the darker the gaps.

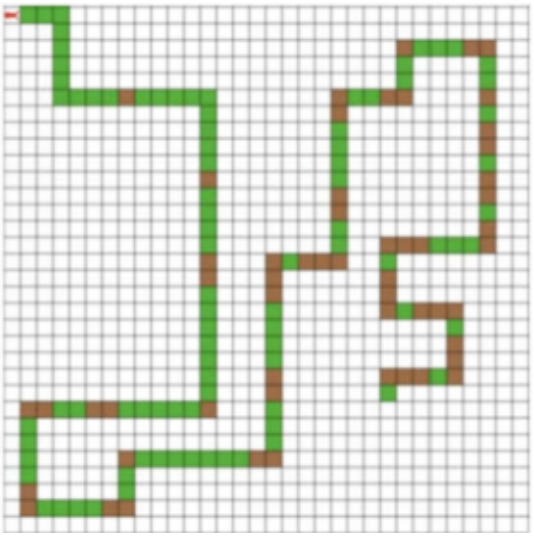

Figure 3.4: The Santa Fe Ant Trail graphical representation.

This trail is placed within an 32x32 toroidal grid (i.e. it does not contain any boundaries, meaning that the top is connected to the bottom and the left connected to the right) and contains 144 cells with 89 pieces of food distributed non-continuously, including 55 gaps and 21 turns. Starting at the upper left corner facing right, the artificial ant can move forward in the direction it is currently facing or turn 90 degrees to the right or left. Each action takes one discrete unit of time to perform. The ant can also perceive if the cell in front of it contains food, an operator that executes instantaneously (i.e. it does not consume any time).

In order to construct the (potential) solutions a program that allows the ant to move and seek for food has to be constructed. Fig.3.5 shows the grammar used in this domain which corresponds to the standard grammar for the Santa Fe Ant Trail in [16].

```
<code> ::= <line> | <code> <line>
<line> ::= <condition> | <op>
<condition> ::= ifelse food-ahead [ <line> ][ <line> ]
<op> ::= turn-left | turn-right | move
```

Figure 3.5: BNF-O'neill grammar definition for the Santa Fe Ant trail.

Each particle repeats its program until all 89 pieces of food are encountered or the maximum number of time-steps is exhausted. Because this number is omitted in O'Neill's experiments in [16], the standard maximum number of steps in GE i.e. 615 was used. For objective-driven search, the traditional fitness function simply counts the units of food eaten by the agent after all time has been exhausted. If the agent eats all foods before time runs out, the problem is considered solved and the ant receives the maximum possible fitness value, which is 89.

For novelty-driven search, two behavior descriptors are applied:

- **behaviorSF1** - A simpler descriptor that adopts the fitness function as the characterization of behavior, as in the Mastermind domain.

- **behaviorSF2** - A more informative characterization which considers the amount of food eaten, with the constraint that the eaten units must not be disconnected from other eaten units along the length of the trail.

In these experiments invalid particles have the fitness of -1 and a ”dummy” behavior descriptor filled with zeros.

For example, in behaviorSF2, if the ant first eats 3 food units that follow the trail, then leaves the trail and eats one more unit in another area of the trail, its behavior descriptor is appended with a 3 (although the ant ate in total 4 units of food), because the last unit is not connected to any other eaten units along the true path of the trail. However, if the ant eats 3 food units at the beginning of the trail, goes off the trail, and collects three more units along a later part of the trail, the score will then be 6. Additionally, this second characterization is sampled over time to provide temporal information about the ant's behavior. In particular, it is sampled every 41 *timesteps*, resulting in a vector of length 15 by the completion of an ant's evaluation.

Note that novelty search coupled with either of these behavior characterizations will not directly search for behaviors that eat more units of food; instead, it will search for novel ways to eat different amounts of food, or in the case of the second characterization, novel time sequences of food eating subject to the continuity constraint.

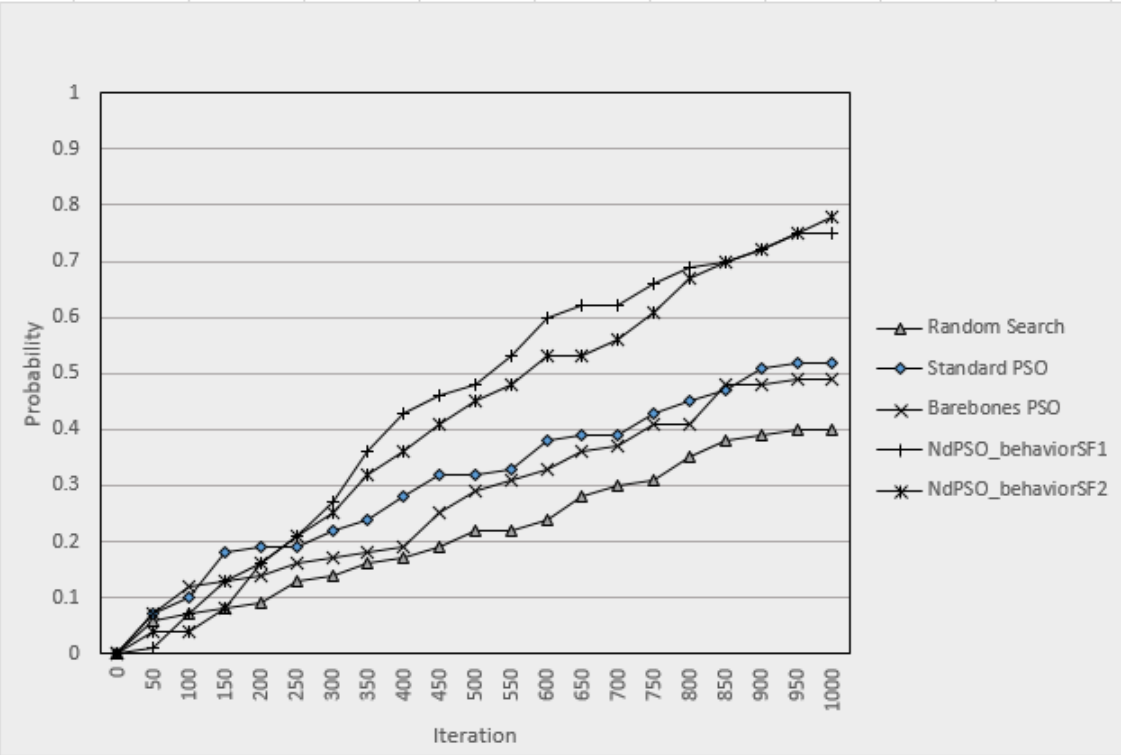

Figure 3.6: The probability for objective-based PSO, Barebones and novelty-driven PSO and random-search to discover solutions in the Stanta Fe Ant Trail.

A plot of the probability of each algorithm to find the solution in the Santa Fe Ant Trail is shown within the 1000 runs in Figure 3.6. The overall performance of objective-driven algorithms is very similar. Both behaviors tested in the NdPSO algorithm outperform Standard PSO and Barebones PSO as well as random-search, this last provides the worse performance of all tested algorithms.

Figure 3.7 compares the average size of the programs obtained with the different algorithms tested. In this domain, NdPSO (behaviorSF1 and behavior SF2) produces significantly smaller programs than Barebones PSO ($p < 0.01$; Student's t-test) and random-search ($p < 0.05$; Student's t-test),. Having smaller programs is important since bloat is a significant problem in GP. However, even though the difference between the length of the problems produced by NdPSO and Standard PSO is not significant, it still has a better performance.

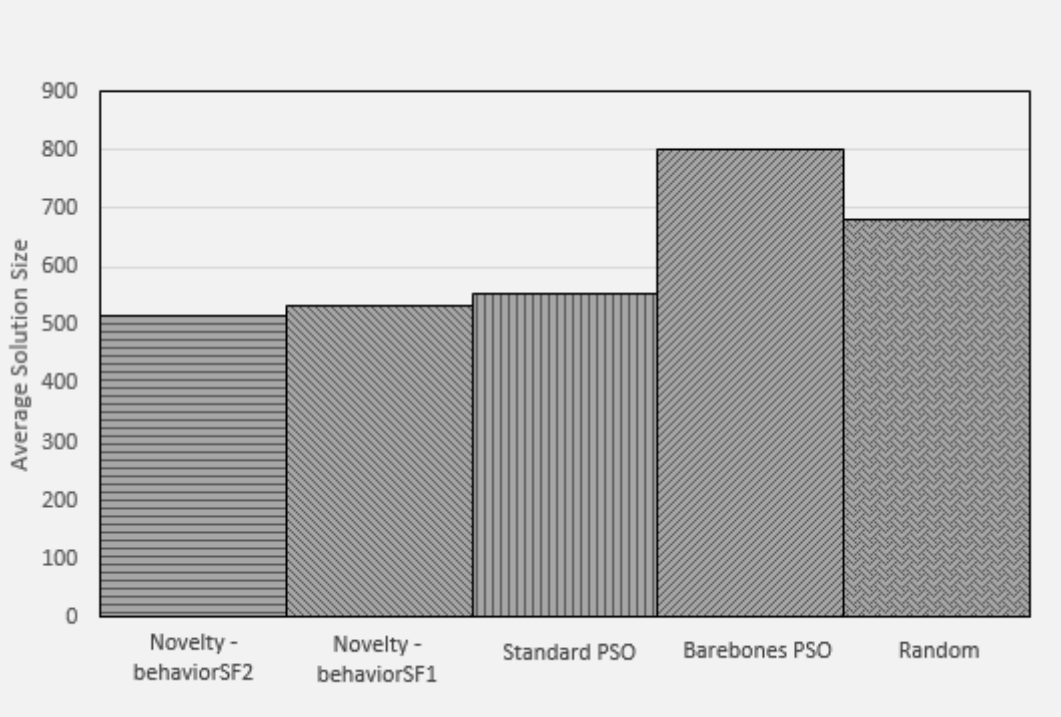

Figure 3.7: Bloat comparison in the Santa Fe Ant Trail domain. Comparison of the average program size (number of characters without spaces) of the best solutions obtained.

3.2.4 Maze Navigation Problem

The Medium map domain is a deceptive and discrete maze navigation task introduced by Lehman and Stanley [9]. The goal in this domain is to find a program that guides an agent in a grid-world domain to the goal location before exhausting the time limit. In these experiments the time limit was set to 500 steps. This maze is suitable for testing GS and NdGS because the placement of the walls create deception. That is, in the Medium map (Figure 3.8), the shortest path to the goal is blocked, such that solving the task requires exploring areas that superficially appear further from the goal.

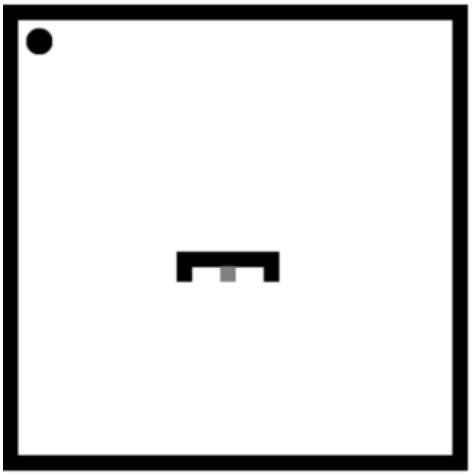

Figure 3.8: Medium map used for the Maze navigation problem.

The possible actions for each agent are: *turn-left*, *turn-right*, *move* and the boolean operators *wall-ahead*, *wall-left* and *wall-right*. In this problem the grammar shown in Fig.3.9 is applied. This was used by Loukas in [71] and Urbano and Loukas in [15].

```
<expr> ::= <line> | <expr> <line>
<line> ::= ifelse <condition> [ <expr> ][ <expr> ] | <op>
<condition> ::= wall-ahead? | wall-left? | wall-right?
<op> ::= turn-left | turn-right | move
```

Figure 3.9: Grammar definition for the Medium Maze problem.

Defining *dist* as the distance from the robot's final position to the goal location, the fitness function for objective-based search is calculated from Equation 3.2.

$$fitness = \frac{1}{1 + dist} \, . \tag{3.2}$$

For novelty search, the behavior descriptor was adopted such that the objective-based search is looking for ways to get closer to the goal, while novelty search instead explores how to reach a diversity of places in the maze:

- **behaviorMP1** - the coordinates of the agent's ending position.

To a particle that generates an invalid program is given a fitness of -1 and a behavior descriptor of a dummy behavior [-100, -100].

Figure 3.10 plots the probability of the solution to be found by the search algorithm during the simulations. NdPSO algorithm outperforms Barebones PSO, Standard PSO and random search.

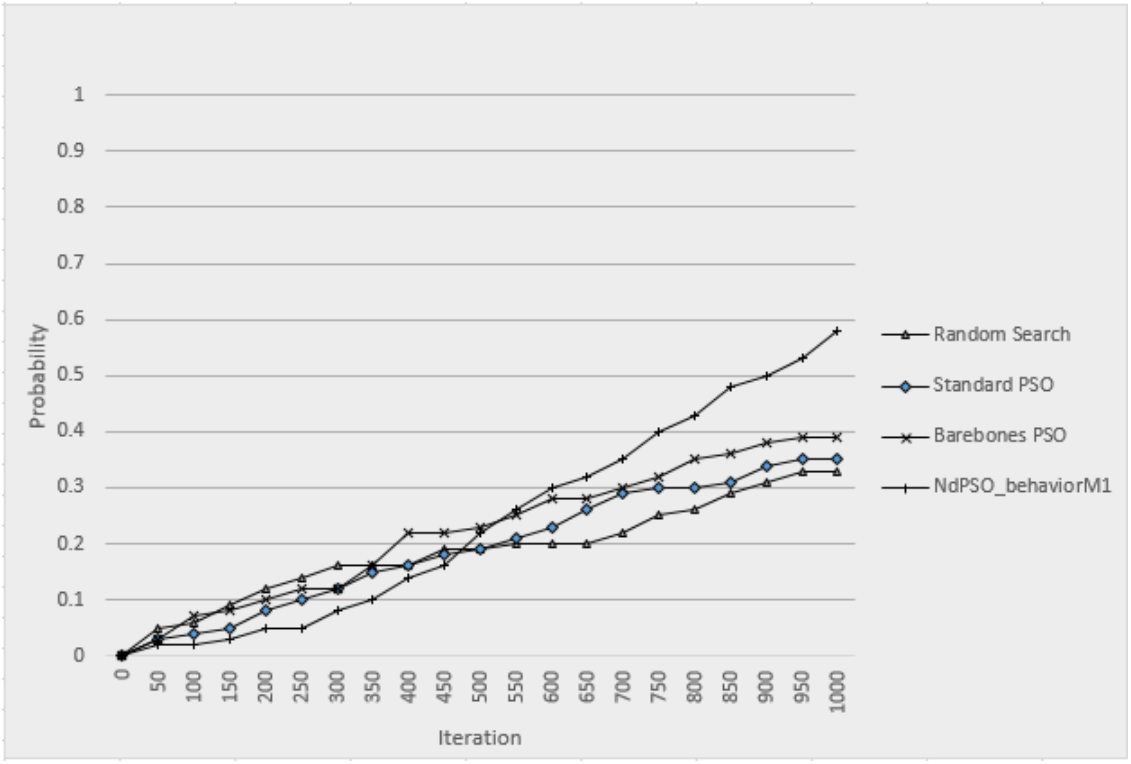

Figure 3.10: The probability for objective-based PSO, Barebones and novelty-driven PSO and random-search to discover solutions in the Maze Navigation Problem.

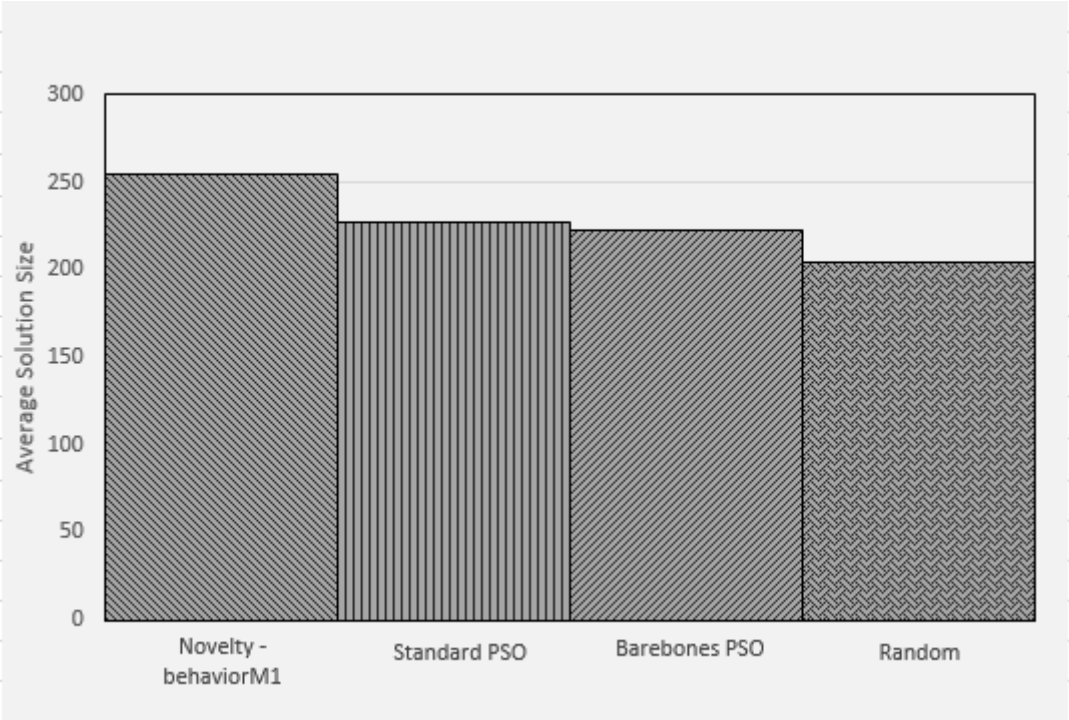

Figure 3.11: Bloat comparison in the Maze navigation domain. Comparison of the average program size (number of characters without spaces) of the best solutions obtained.

The average program length is presented in Figure 3.11. A Student's t-test statistical test with a significance level of 0.05 shows that in this domain NdPSO produces significantly larger programs that the other tested algorithms.

3.2.5 Main Results

To tune the performance of the algorithms, many combinations of topologies and other non-fixed parameters were tested in all the domains. Namely, the imposition of only valid programs throughout the experiments and, in the specific case of the NdPSO, the best value for *k-nearest-distances* and the existence or not of an archive of past behaviors. Due to its volume, those tests and its results for the Mastermind, Santa Fe Ant Trail and the Maze Navigation Problem are presented in sections A.1, A.2 and A.3 of Appendix A, respectively. Table B.4 provides a summary of the best results obtained for each algorithm in the considered domain.

Table 3.2: Comparison of the results obtained for PSO, NdPSO and random-search averaged over 100 runs.

	Topology	Mean best fitness	Std Deviation	Median	Successful runs
Mastermind					
PSO	Fully-connected	0.90	0.04	0.89	14
Barebones PSO	Ring	0.87	0.04	0.89	2
NdGS-behaviorMm1	Ring	0.92	0.05	0.89	25
NdPSO-behaviorMm2	Ring	**1.00**	**0.00**	**1.00**	**100**
Random	N/A	0.86	0.05	0.89	0
Santa Fe Trail					
PSO	Fully-connected	78.31	15.65	89.0	52
Barebones PSO	Fully-connected	74.25	16.95	88.0	49
NdPSO-behaviorSF1	Ring	85.46	8.54	89.0	75
NdPSO-behaviorSF2	Ring	**87.89**	**6.52**	**89.0**	**78**
Random	N/A	75.81	15.74	88.0	40
Maze problem					
PSO	Von Neumann	0.50	0.37	0.31	35
Barebones PSO	Fully-connected	0.52	0.39	0.31	39
NdPSO-behaviorMP1	Ring	**0.68**	**0.38**	**1.00**	**58**
Random	N/A	0.47	0.38	0.24	33

3.2.6 Discussion

Contradicting previous work that revealed best performance when using Barebones PSO instead of Standard PSO [6, 30, 31], in these experiments almost the exact opposite occurred. Applying Barebones PSO to GS did not improve the results. This method performed poorly in all domains, and even provided significantly worse results than Standard PSO in Mastermind and the Santa Fe Ant Trail.

It is also important to note that there is not a singular best topology over all for all experimental setups. In general, the best results obtained in objective-driven PSO (with and without Barebones PSO) used the Fully-connected topology, where in NdPSO the Ring topology provided substantially improved results when compared with the others tested.

Generally, if particles are forced to generate valid programs, the algorithms' performance improves considerably.

Regarding Mastermind in the NdPSO implementation, the archive considerably degrades the performance of the algorithm, because the algorithm had very few success in the 100 tests for all the combinations tested. The reason is that in this domain the number of possible behaviors is relatively small and thus many particles will have the same behavior. A similar outcome resulted in the Santa Fe Trail when using behaviorSF1 as a behavior descriptor, because in this case only 89 unique behaviors exist. Because of the way that novelty is calculated (Equation 2.8), if the number of particles with the same behavior is equal or bigger than the k-nearest-distances considered, which in this case appears many times, the novelty of all those particles will be 0 (the distance between equal behaviors is 0). Consequently, many particles will have the same behavior and the most novel has no meaning, in this case the algorithm will degrade into random search. However, when using behaviorSF2 in Santa Fe Trail and in the Maze Navigation problem, an archive of past behaviors produced better results.

Relatedly, testing different behavior descriptors when applying novelty search is often helpful. In these experiments, choosing a more complex descriptor, in particular a descriptor other than the fitness score, helped improve performance. In the Mastermind problem, considered a difficult problem for GP algorithms, when using a more complex behavior descriptor NdPSO managed to succeed in all runs, a considerable improvement comparing to the more simpler descriptor provided 25% of successes, and objective-driven GS achieved success in only 14% of runs.

In the Santa Fe Ant Problem domain, NdPSO produces significantly smaller programs than the other algorithms, still outperforming them. This is an important result because it means that, comparatively to the others, NdPSO is more effective computationally and smaller programs are more generalizable and more intuitive to one to comprehend. In the other domains, on average, NdPSO creates larger programs, but using a Pearson's Correlations Test it is evident that a correlation exists between the average size of a program and its probability of success. With 95% of confidence, the Pearson's Correlations Test calculates a correlation coefficient of 0.50 for the Mastermind problem and 0.92 for the Maze navigation problem, implying that it is unlikely to obtain with smaller programs a higher percentage of success than the one obtained by NdPSO. However more testing is necessary to determine if there is still some bloat with NdPSO and smaller programs could obtain equal performance.

In all problems NdPSO outperforms the objective-based algorithms tested, both in number of solutions found and in the best fitnesses discovered averaged over all runs.

3.3 Conclusion

This chapter introduced Novelty-driven PSO (NdPSO), a method that aims to overcome the significant challenge of premature convergence in traditional objective-driven PSO. NdPSO was coupled to the Grammatical Evolution (GE) mapping process, to generate and evolve problems in an arbitrary language. This method was tested in three domains, where in all it outperformed the objective-based PSO, Barebones PSO and the random-search. In this way, NdPSO shows promise for solving deceptive PSO problems and encourages further follow-up investigation. Next, NdPSO will be compared to NdGE (i.e. Novelty driven GE), the evolutionary method nearest to it in approach.

Chapter 4

Novelty-driven PSO and Novelty Search Comparative Study

An empirical study comparing the algorithm developed - NdPSO - to the novelty search standard implementation as a Evolutionary Algorithm is presented in this chapter. The following section introduces the aim of this chapter followed by a theoretical comparison of a typical Genetic Algorithm (GA) to Particle Swarm Optimization (PSO). Next, the necessary alterations to Grammatical Evolution (GE) in order to make it novelty-driven are listed and explained. Section 4.4 covers the experiments that have been made with novelty-driven GE: the experimental parameters and the results of the experiments are presented. The chapter ends with a discussion about the obtained results and a conclusion.

4.1 Introduction

Novelty search was first introduced in Evolutionary Computation as a method to mitigate deception for Evolutionary Algorithms. In the previous chapter, novelty search was extended to PSO, and the performance of objective-driven PSO and the novelty-driven one was compared in three different benchmarks. In all of the benchmarks, NdPSO was able to outperform the Standard PSO, the Barebones PSO, both objective-driven, and the random-search. By outperforming the objective-driven version, the previous results proved that NS is a technique that can be successfully transferred to population-based optimization algorithms outside the evolutionary paradigm.

This chapter's main objective is to compare the novelty search implemented with PSO (NdPSO) to the canonical evolutionary novelty-search algorithm. This study is important because it analyses the competitiveness of the developed algorithm across search paradigms.

4.2 Genetic Algorithm and Particle Swarm Optimization Comparison

In the following sections, a comparison of the results obtained with NS, which is ultimately a GA, and NdPSO will be made. For this reason the theoretical differences and similarities between these two methods will now be described.

PSO is similar to the Genetic Algorithm approach in the sense that it also is a population-based method and it relies on information-sharing among their population members. As mentioned earlier, a PSO particle, characterized by its position and velocity, is comparable to an individual or chromosome in a GA. They both represent candidate solutions to the designated problem.

The main difference between PSO and a GA is in how population members evolve towards a goal. In PSO the particles update their positions in order to optimize a fitness function. In contrast, in a GA a new generation of individuals is generated from the previous one through genetic operators, like selection, crossover and mutation (a review of these operators is provided in section 2.2.1). Even though PSO does not use such operators, analogies between genetic operators and other processes in PSO exist.

For example, the use of crossover in a GA exchanges genetic material between individuals. This helps propagate good features between generations so that future generations will be more fit and overall the simulation will converge more quickly. While PSO does not apply crossover, an analog of this concept is realized through how particles exert influence on the trajectories of other particles. That is, by being attracted by its own best position and the best position found by the whole swarm (or best position found by its neighborhood when using neighborhood topologies) the particles will move to higher fitness areas and eventually converge.

The mutation operator in a GA introduces randomness in the population, making it theoretically possible for an individual to reach any place in the search space if the mutation rates are high enough. However this is unlikely to happen in practice, because as the generation's average fitness becomes higher, applying mutation will increasingly result in lower-fitness individuals that have low probability of persisting through selection. In PSO, each particle can reach any point in space using only one step but only in the beginning of the simulation, and only if v_{max} is sufficiently large. However, particles can technically reach any place in the search space over a number of iterations, because they are not replaced.

A GA uses selection to pick fitter individuals and guarantee fitness survival. Hence, individuals with worse fitness will have higher probability to be removed from the population in order to, hopefully, generate a fitter offspring. PSO does not incorporate a selection technique, also, in PSO there is no concept of generations because the initial particles are not overwritten during optimization: only their position changes over time.

Therefore, there is not a direct analogy between the selection process in EAs and PSO, leaving a loose end where this method can be improved. Hence many hybrid algorithms of PSO that couple GA and PSO have been proposed [72, 73].

4.3 Novelty Search applied to Grammatical Evolution

In the previous section a comparison between a GA and PSO was made, but recall that the goal is to compare the evolutionary NS to NdPSO. But to fairly make this comparison a grammatical approach will also be implemented to the standard NS algorithm. In this section this implementation will be described.

Tracking novelty requires small changes to GE besides replacing the fitness function by a novelty metric and adding an archive. As before, in this study the Euclidean distance between the behavior descriptors is used as a novelty metric, the behavior descriptors depend on the benchmark domain.

Most commonly, GE is implemented with some kind of objective-based search mechanism. This way, the most fitter individuals will have higher chances of being present in the nest generations propagating their genetic material. When implementing GE with NS the selection process is novelty dependent. Meaning that instead of individuals with better fitness, the selection process will pick individuals with higher novelty score.

4.4 Experiments

As the previous ones, all of the experiments bellow have been performed in Netlogo [67], using a Java implementation of the GE algorithm [68] - the jGE library - and the respective Netlogo extension [69] - jGE.

The aim of these experiments is to compare the performance of NdPSO proposed in this thesis, with the performance of novelty search inside the Evolutionary Computation paradigm, where it was implemented originally. The algorithms were tested in three benchmark domains (the same as in the previous chapter's experiments): the Mastermind problem (Section 3.2.2), the Santa Fe Ant Trail (Section 3.2.3) and a Maze Navigation problem (Section 3.2.4).

Note that because NdPSO was implemented as an extension of Grammatical Evolution, in this study, the evolutionary novelty search is also implemented combined with GE. Although it is not part of this chapter's objective, objective-driven GE was also tested.

Both objective-driven and novelty-driven GE where tested without the *always valid* parameter, this way, in order to make a fair comparison, the results of novelty-driven PSO to which the latter are compared are not the best results obtained but the ones from the experiments also without *always valid*.

4.4.1 Experimental Settings

Over these experiments, some parameters were fixed in order to study the impact of others. For the NdPSO the same set of experiments is used in these comparison, this way, the parameters used are presented in the Table3.1. For the evolutionary GE implementation of novelty search, the parameters used by O'Neill for GE in his first GS experiments [16] are used. As he explained, these parameters are chosen in order to achieve a relatively fair comparison between the different approaches. To that, the same number of evaluations was maintained and the most common population sizes in the literature for GE and PSO were adopted. Table 4.1 presents the values of all fixed parameters used for the evolutionary novelty search implementation throughout these experiments. A succinct description of some less-intuitive parameters is presented next.

Table 4.1: Fixed parameters used on the GE novelty search implementation for its comparison to NdPSO.

Parameter	Value
Initial population	random
Population-size	500
Chromosome length	100
Max-Generations	60
Codon-size	8
Max-wraps	10
Generation gap	90%
Crossover	90%
Codon mutation	1%

Generation gap It is very common in EAs to have some type of elitism where fitter individuals are preserved in the following generation. This parameter determines the percentage of the population that is replaced in the next generation.

Chromosome length As a consequence of the crossover process, most often, the length of the chromosomes can increase or decrease during the experiments. In order to fairly compare the evolutionary implementation of NS to NdPSO, since this last uses fixed length position vectors, in this implementation the chromosome length is also fixed. Thus, if the length of the chromosome exceeds the set value, the resulting integer vector is then cut in order to have the desired length.

Selection method Evolutionary algorithms use a selection algorithm in order to pick the fitter individuals and place them into a mating pool. As did O'Neill [16], in these experiments the roulette wheel selection method was used. This way, individuals with better fitnesses will have higher probability of being picked. Although less fitted individuals will be, most likely, eliminated from the next generation, there is still a chance they might not be.

Crossover method A One-Point Crossover technique was used. This means that the ”parents” chromosome is cut in only one place, and then combined to create the children.

4.4.2 Results

In novelty-driven GE many combinations of *k-nearest-distances* were tested as well as the inclusion or not of an archive of past behaviors in all domains. These tests allowed to perform some tuning on the algorithm's performance the same way it was done in the previous experiments. The results of all experiments can be observed in Appendix B.2. Regarding objective-driven GE, these experiments were not performed since there is no notion of *k-nearest-distances* or archive.

Table 4.2: Comparison of the results obtained for GE, novelty-driven GE and NdPSO over 100 runs.

	Archived behaviors	Mean best fitness	Std Deviation	Median	Successful runs
Mastermind					
GE	NA	0.90	0.04	0.89	14
Novelty-driven GE	FALSE	0.90	0.04	0.89	20
Novelty-driven PSO	**FALSE**	**0.91**	**0.05**	**0.89**	**22**
Santa Fe Trail					
GE	NA	79.61	14.60	89	64
Novelty-driven GE	**TRUE**	**87.64**	**4.75**	**89**	**90**
Novelty-driven PSO	TRUE	82.82	11.03	89.0	62
Maze problem					
GE	NA	0.87	0.30	1.00	83
Novelty-driven GE	**FALSE**	**1.00**	**0.00**	**1.00**	**100**
Novelty-driven PSO	FALSE	0.35	0.34	0.17	20

4.5 Discussion

Unlike in NdPSO, where the inclusion of an archive of past behaviors in the Mastermind benchmark domain resulted in poor results, in novelty-driven GE, although it did not improve the performance of the algorithm when compared with its version without the archive, it did not degrade the performance as dramatically. The reason is likely because NdPSO is implemented in such a way that, when the novelty of *gnovel* does not improve, *gnovel* will remain the same. So when using an archive of past behaviors, early on the simulation all particles will have minimum novelty, and will keep approaching the same behavior. As particles proceed to follow the most novel, and because it will move very little, since it will follow itself, it will be progressively unlikely that one particle moves to

a position that will be mapped into a successful program. However, in GE (and novelty-driven GE), even though individuals will soon all have minimum novelty, because of the intrinsic randomness of the crossover process there is still a considerable chance of creating a genotype that will be mapped into a successful program.

The Medium map seems to be a fairly easy problem for the Evolutionary Algorithms (note that, even objective-driven GE had a much higher performance than novelty-driven PSO). In the future, it would be interesting to test both Novelty-driven PSO and novelty-driven GE in a different maze in order to infer if the performance's discrepancy will remain proportional.

Regarding the performance comparison between novelty-driven GE and Novelty-driven PSO, NdPSO was able to outperform the evolutionary implementation of novelty search in one out of three benchmark domains. Even though NdPSO performance is not outstanding when compared with novelty-driven GE, it still shows potential because the number of particles (30) in NdPSO is much smaller than the number of individuals used in the evolutionary experiments (500). Recall that the number of individuals in GE and novelty-driven GE was chosen in order to be concordant to the ones often reported in the state of the art.

4.6 Conclusion

This chapter compared Novelty-driven PSO to canonical novelty search (implemented as a Genetic Algorithm). In both cases, a grammatical approach was used: NdPSO was coupled to the Grammatical Evolution mapping process and novelty search was implemented on top of GE, resulting in a novelty-driven GE. This comparative study was made in three domains, where in one of them (the Mastermind problem) NdPSO outperformed novelty-driven GE. Although it did not outperform novelty-driven GE in two out of three problems, there still exists room for improvement. Because NdPSO has a considerable less amount of parameters than the Evolutionary Algorithms, making it more intuitive, easier to implement and tune, NdPSO proves again to be a promising algorithm.

Chapter 5

Conclusion

The hypothesis of this dissertation was to developed a model that can avoid premature convergence to local optima in Particle Swarm Optimization by guiding the search towards novel behaviors instead of the objective. I accomplished this by combining novelty search (a fairly recent technique) to the PSO algorithm and create a new method called Novelty-driven Particle Swarm Optimization (Novelty-driven PSO or NdPSO). I implemented NdPSO as an extension of the Grammatical Swarm which is PSO-based version of Grammatical Evolution that is used to evolve programs in arbitrary languages.

After implementing the algorithm, I tested NdPSO and compared it to objective-driven PSO and the random-search in three different difficult and deceptive domains. The fist domain, the Mastermind problem, is a code breaker game where discovering a hidden pin sequence is the objective. This is a very deceptive problem because of the way in which the fitness function was defined. The other two domains (the Santa Fe Ant Trail and the Maze Navigation Problem) are deceptive reinforcement learning benchmarks commonly used in Genetic Programming.

Novelty search is commonly implemented in Evolutionary Algorithms which differs from PSO because it use techniques like selection, crossover and mutation to mimic the biological evolution. So, after demonstrating that Novelty-driven PSO can outperform objective-driven PSO, I compared the former to a more common evolutionary implementation of novelty search in the same three benchmarks used before. The goal was to infer if NdPSO is a competitive search algorithm compared with evolutionary novelty search.

The results showed that in all the benchmarks NdPSO greatly outperformed the best performance achieved by objective-based standard PSO, Barebones PSO and the random-search, improving the number of times the Mastermind was successfully completed by 86 percentage points (*p.p.*) from standard objective-driven PSO, 98*p.p.* from Barebones PSO and 100*p.p.* from random search; in the Santa Fe Ant Trail NdPSO concluded successfully within the limit of time-steps more 26*p.p.* than standard PSO, 29*p.p.* and 38*p.p.* than Barebones PSO and random-search respectively; in the Maze Navigation problem NdPSO also improved by 23, 19 and 25*p.p.* compared to standard PSO, Barebones PSO and

random-search, respectively. Based on these results, a paper was submitted and accepted for the Biennial International Conference on Artificial Evolution (EA-2015) that will take place from 26 to 28 October 2015 in Lyon, France.

When compared to the evolutionary novelty search, NdPSO was able to outperform it in the Mastermind problem, staying behind in the other two. This can be due to the fact that the number of particles used in NdPSO is much smaller than the number of individuals used in the evolutionary experiments. However, the fact that NdPSO inherited from PSO one of its advantages, the reduced number of parameters, makes it more intuitive than Evolutionary Algorithms and easier to implement and tune. Overall, NdPSO shows promise for solving deceptive PSO problems and it has room for improvement encouraging further follow-up investigation.

5.1 Future Work

The work developed in this dissertation showed that the proposed method is very promising for solving deceptive problems in PSO. However, the scope of this project and the time constraints was limited to developing the method and proving its viability. This way, some phenomena was registered to which many explanations or no explanation can be provided. For example, one open question is why the results do not improve in the objective-based search with the Ring and Von Neumann topologies but do when the search is novelty-driven. Theoretically, these neighborhood topologies should delay premature convergence, but this intuition does not hold up. Further investigation would help to better understand the method's functioning.

Although NdPSO performed better than objective-driven PSO, the proposed algorithm can still be improved specially when compared with the EA version of Novelty Search. Recently, some issues related with GE were pointed out by some authors, such as low locality and high levels of redundancy [74, 75]. A possible solution to low locality, which is when small modifications in the genotype have a huge effect in phenotype, has been proposed. This method, Structured Grammatical Evolution (SGE) [76], is an adaptation of GE, where the linear genotype is replaced by a structured one, where each gene is a list of integers that represent the possible derivation choices of non-terminals, thereby increasing locality. One possible improvement to NdPSO could be using a different implementation where SGE is coupled with PSO, similar to what was done with GS, and then adding the novelty paradigm.

Appendix A

Results of the objective-driven and novelty-driven PSO comparison

The following sections present the results of the experiments carried out for the comparison of objective-driven PSO, objective-driven Barebones PSO, random-search and novelty-driven PSO.

A.1 Results of the Mastermind tests

Table A.1: Results for the Mastermind problem using Standard PSO, Barebones PSO and random-search algorithms. Each result represents a different experiment composed by 100 runs. $\bar{x}$, σ, $\tilde{x}$ represents, respectively, the average, standard deviation and median of the best fitnesses of each run. $\%$ success - represents the number of success runs on 100 possible.

Method	Always valid	Topology	$\bar{x}$	σ	$\tilde{x}$	$\%$ success
Standard PSO	FALSE	Fully-connected	0.90	0.04	0.89	14
		Von Neumann	0.90	0.03	0.89	8
		Ring	0.89	0.02	0.89	4
	TRUE	**Fully-connected**	**0.90**	**0.04**	**0.89**	**14**
		Von Neumann	0.90	0.03	0.89	11
		Ring	0.90	0.03	0.89	10
Barebones PSO	FALSE	Fully-connected	0.89	0.02	0.89	0
		Von Neumann	0.88	0.04	0.89	1
		Ring	0.88	0.04	0.89	1
	TRUE	Fully-connected	0.88	0.03	0.89	1
		Von Neumann	0.87	0.04	0.89	0
		Ring	**0.87**	**0.04**	**0.89**	**2**
random-search	FALSE	N/A	0.86	0.05	0.89	0
	TRUE	N/A	0.86	0.05	0.89	0

Table A.2: Results for the Mastermind problem using the Novelty-driven PSO, particularly, the simpler behavior **behaviorMm1**. Each result represents a different experiment composed by 100 runs. $\bar{x}$, σ, $\tilde{x}$ represents, respectively, the average, standard deviation and median of the best fitnesses of each run. $\%$ success - represents the number of success runs on 100 possible.

	Always valid	Topology	$\bar{x}$	σ	$\tilde{x}$	$\%$ success
k-distances = 3	FALSE	Fully-connected	0.91	0.04	0.89	15
		Von Neumann	0.91	0.04	0.89	17
		Ring	0.91	0.04	0.89	15
	TRUE	Fully-connected	0.90	0.03	0.89	10
		Von Neumann	0.91	0.04	0.89	19
		Ring	0.10	0.04	0.89	16
k-distances = 10	FALSE	Fully-connected	0.91	0.04	0.89	15
		Von Neumann	0.91	0.05	0.89	22
		Ring	0.91	0.04	0.89	19
	TRUE	Fully-connected	0.90	0.03	0.89	9
		Von Neumann	0.91	0.05	0.89	21
		Ring	0.91	0.05	0.89	23
k-distances = 15	FALSE	Fully-connected	0.91	0.05	0.89	21
		Von Neumann	0.90	0.04	0.89	15
		Ring	0.91	0.05	0.89	23
	TRUE	Fully-connected	0.90	0.04	0.89	13
		Von Neumann	0.91	0.05	0.89	21
		Ring	**0.92**	**0.05**	**0.89**	**25**
k-distances = 25	FALSE	Fully-connected	0.90	0.04	0.89	14
		Von Neumann	0.91	0.04	0.89	15
		Ring	0.89	0.05	0.91	22
	TRUE	Fully-connected	0.91	0.04	0.89	17
		Von Neumann	0.91	0.05	0.89	22
		Ring	0.91	0.04	0.89	16

Table A.3: Results for the Mastermind problem using the Novelty-driven PSO, particularly, the more complex behavior **behaviorMm2**. Each result represents a different experiment composed by 100 runs. $\bar{x}$, σ, $\tilde{x}$ represents, respectively, the average, standard deviation and median of the best fitnesses of each run. $\%$ success - represents the number of success runs on 100 possible.

	Always valid	Topology	$\bar{x}$	σ	$\tilde{x}$	$\%$ success
k-distances = 3	FALSE	Fully-connected	0.96	0.05	1.00	60
		Von Neumann	0.99	0.03	1.00	94
		Ring	1.00	0.02	1.00	97
	TRUE	Fully-connected	0.96	0.05	1.00	65
		Von Neumann	1.00	0.02	1.00	98
		Ring	1.00	0.02	1.00	96
k-distances = 10	FALSE	Fully-connected	0.94	0.06	0.94	50
		Von Neumann	1.00	0.02	1.00	98
		Ring	1.00	0.02	1.00	98
	TRUE	Fully-connected	0.96	0.05	1.00	62
		Von Neumann	1.00	0.01	1.00	99
		Ring	1.00	0.01	1.00	99
k-distances = 15	**FALSE**	Fully-connected	0.95	0.06	1.00	56
		Von Neumann	1.00	0.01	1.00	99
		Ring	**1.00**	**0.00**	**1.00**	**100**
	TRUE	Fully-connected	0.95	0.05	1.00	59
		Von Neumann	1.00	0.01	1.00	99
		Ring	1.00	0.01	1.00	99
k-distances = 25	FALSE	Fully-connected	0.94	0.06	0.89	49
		Von Neumann	1.00	0.02	1.00	99
		Ring	1.00	0.01	1.00	99
	TRUE	Fully-connected	0.95	0.06	1.00	58
		Von Neumann	**1.00**	**0.00**	**1.00**	**100**
		Ring	**1.00**	**0.00**	**1.00**	**100**

A.2 Results of the Santa Fe Ant Trail tests

Table A.4: Results for the Santa Fe Ant Trail using Standard PSO, Barebones PSO and random-search algorithms. Each result represents a different experiment composed by 100 runs. $\bar{x}$, σ, $\tilde{x}$ represents, respectively, the average, standard deviation and median of the best fitnesses of each run. $\%$ success - represents the number of success runs on 100 possible.

Method	Always valid	Topology	$\bar{x}$	σ	$\tilde{x}$	$\%$ success
Standard PSO	FALSE	Fully-connected	71.30	17.37	71.0	31
		Von Neumann	77.03	16.19	88.0	44
		Ring	73.34	16.80	76.0	34
	TRUE	**Fully-connected**	**78.31**	**15.66**	**89.0**	**52**
		Von Neumann	75.01	16.01	88.0	40
		Ring	76.94	15.71	88.0	46
Barebones PSO	FALSE	Fully-connected	71.77	15.79	71.0	34
		Von Neumann	69.34	17.38	66.0	30
		Ring	68.33	17.25	66.0	26
	TRUE	**Fully-connected**	**74.25**	**16.95**	**88.0**	**49**
		Von Neumann	71.98	16.05	71.0	31
		Ring	71.23	15.86	71.0	29
random-search	FALSE	N/A	73.00	16.68	80.5	31
	TRUE	**N/A**	**75.81**	**15.74**	**88.0**	**40**

Table A.5: Results for the Santa Fe Ant Trail using the Novelty-driven PSO, particularly, the simpler behavior **behaviorSF1**, not archiving past behaviors. Each result represents a different experiment composed by 100 runs. $\bar{x}$, σ, $\tilde{x}$ represents, respectively, the average, standard deviation and median of the best fitnesses of each run. $\%$ success - represents the number of success runs on 100 possible.

	Always valid	Topology	$\bar{x}$	σ	$\tilde{x}$	$\%$ success
k-distances = 3	FALSE	Fully-connected	78.99	14.19	88.0	47
		Von Neumann	77.66	14.91	88.5	50
		Ring	82.13	12.10	89.0	58
	TRUE	Fully-connected	81.10	14.14	89.0	65
		Von Neumann	82.96	11.84	89.0	66
		Ring	**85.46**	**8.54**	**89.0**	**75**
k-distances = 10	FALSE	Fully-connected	77.37	15.18	88.0	41
		Von Neumann	72.34	16.25	71.0	37
		Ring	78.56	13.95	88.0	42
	TRUE	Fully-connected	78.8	14.11	88.0	48
		Von Neumann	79.08	14.70	88	43
		Ring	79.36	14.65	89.0	57
k-distances = 15	FALSE	Fully-connected	73.43	16.71	76.0	35
		Von Neumann	72.93	16.26	73.0	35
		Ring	75.95	16.88	88.0	48
	TRUE	Fully-connected	77.41	16.09	88.0	48
		Von Neumann	77.54	15.11	88.0	44
		Ring	79.09	14.19	88.0	49
k-distances = 25	FALSE	Fully-connected	71.38	17.66	71.0	38
		Von Neumann	73.79	16.83	88.0	38
		Ring	74.42	16.46	85.0	38
	TRUE	Fully-connected	74.44	16.18	85.0	40
		Von Neumann	78.49	14.40	88.0	47
		Ring	75.26	16.46	88.0	39

Table A.6: Results for the Santa Fe Ant Trail using the Novelty-driven PSO, particularly, the simpler behavior **behaviorSF1**, archiving past behaviors. Each result represents a different experiment composed by 100 runs. $\bar{x}$, σ, $\tilde{x}$ represents, respectively, the average, standard deviation and median of the best fitnesses of each run. $\%$ success - represents the number of success runs on 100 possible.

	Always valid	Topology	$\bar{x}$	σ	$\tilde{x}$	$\%$ success
k-distances = 3	FALSE	Fully-connected	77.15	16.58	89.0	53
		Von Neumann	78.14	14.51	89.0	51
		Ring	83.00	11.32	89.0	58
	TRUE	Fully-connected	75.1	16.67	88.5	50
		Von Neumann	82.68	11.48	89.0	61
		Ring	**83.83**	**10.76**	**89.0**	**66**
k-distances = 10	FALSE	Fully-connected	72.83	16.67	73.5	39
		Von Neumann	76.62	15.36	89.0	51
		Ring	82.82	11.03	89.0	62
	TRUE	Fully-connected	77.9	15.16	89.0	51
		Von Neumann	81.99	12.14	89	55
		Ring	83.48	10.82	89.0	65
k-distances = 15	FALSE	Fully-connected	76.32	16.37	88.0	43
		Von Neumann	78.49	14.40	88.0	47
		Ring	77.70	15.27	88.0	49
	TRUE	Fully-connected	75.83	16.20	88.0	49
		Von Neumann	82.91	11.78	89.0	59
		Ring	82.12	11.43	89.0	60
k-distances = 25	FALSE	Fully-connected	74.80	16.41	88.0	43
		Von Neumann	80.12	14.33	89.0	57
		Ring	80.33	12.98	89.0	53
	TRUE	Fully-connected	79.71	13.23	88.0	49
		Von Neumann	81.07	13.23	89.0	58
		Ring	84.55	10.25	89.0	71

Table A.7: Results for the Santa Fe Ant Trail using the Novelty-driven PSO, particularly, the behavior **behaviorSF2**, not archiving past behaviors. Each result represents a different experiment composed by 100 runs. $\bar{x}$, σ, $\tilde{x}$ represents, respectively, the average, standard deviation and median of the best fitnesses of each run. $\%$ success - represents the number of success runs on 100 possible.

	Always valid	Topology	$\bar{x}$	σ	$\tilde{x}$	$\%$ success
k-distances = 3	FALSE	Fully-connected	79.72	14.52	89.0	54
		Von Neumann	78.42	15.69	89.0	55
		Ring	80.95	14.05	89.0	57
	TRUE	Fully-connected	81.92	13.35	89.0	62
		Von Neumann	81.59	12.71	89.0	55
		Ring	**83.34**	**11.94**	**89.0**	**69**
k-distances = 10	FALSE	Fully-connected	73.14	15.96	74.5	31
		Von Neumann	73.70	16.88	86.5	41
		Ring	76.06	16.61	88.0	48
	TRUE	Fully-connected	73.86	16.55	80.5	37
		Von Neumann	74.26	15.83	80.5	33
		Ring	76.18	15.94	88.0	44
k-distances = 15	FALSE	Fully-connected	73.87	16.97	88.0	38
		Von Neumann	73.21	16.93	85.0	40
		Ring	74.05	16.66	85.0	40
	TRUE	Fully-connected	76.99	14.98	88.0	47
		Von Neumann	75.85	15.55	86.5	38
		Ring	75.53	15.08	88.0	35
k-distances = 25	FALSE	Fully-connected	74.57	16.53	88.0	41
		Von Neumann	75.21	16.80	88.0	39
		Ring	70.34	18.08	71.0	34
	TRUE	Fully-connected	77.33	15.67	89.0	51
		Von Neumann	76.34	15.95	88.0	46
		Ring	70.37	16.52	71.0	30

Table A.8: Results for the Santa Fe Ant Trail using the Novelty-driven PSO, particularly, the behavior **behaviorSF2**, archiving past behaviors. Each result represents a different experiment composed by 100 runs. $\bar{x}$, σ, $\tilde{x}$ represents, respectively, the average, standard deviation and median of the best fitnesses of each run. $\%$ success - represents the number of success runs on 100 possible.

	Always valid	Topology	$\bar{x}$	σ	$\tilde{x}$	$\%$ success
k-distances = 3	FALSE	Fully-connected	73.97	17.15	85.0	35
		Von Neumann	78	15.03	89.0	51
		Ring	80.21	13.58	89.0	58
	TRUE	Fully-connected	81.07	12.84	89.0	54
		Von Neumann	83.15	10.67	89.0	62
		Ring	**87.09**	**6.52**	**89.0**	**78**
k-distances = 10	FALSE	Fully-connected	76.90	15.83	88.0	46
		Von Neumann	79.29	14.73	88.0	46
		Ring	81.61	12.37	89.0	57
	TRUE	Fully-connected	78.45	14.66	89.0	52
		Von Neumann	82.68	12.37	89.0	60
		Ring	83.43	11.29	89.0	65
k-distances = 15	FALSE	Fully-connected	78.60	14.14	88.0	48
		Von Neumann	77.27	15.15	88.0	45
		Ring	80.01	13.58	89.0	51
	TRUE	Fully-connected	77.28	15.62	88.0	46
		Von Neumann	80.56	13.10	89.0	56
		Ring	83.25	11.14	89.0	66
k-distances = 25	FALSE	Fully-connected	77.72	15.64	88.5	50
		Von Neumann	78.25	15.34	88.0	49
		Ring	76.42	15.16	88.0	43
	TRUE	Fully-connected	79.63	14.13	89.0	51
		Von Neumann	83.23	11.71	89.0	62
		Ring	80.36	13.25	88.5	50

A.3 Results of the Maze Navigation Problem tests

Table A.9: Results for the Maze Navigation Problem using Standard PSO, Barebones PSO and random-search algorithms. Each result represents a different experiment composed by 100 runs. $\bar{x}$, σ, $\tilde{x}$ represents, respectively, the average, standard deviation and median of the best fitnesses of each run. $\%$ success - represents the number of success runs on 100 possible.

Method	Always valid	Topology	$\bar{x}$	σ	$\tilde{x}$	$\%$ success
Standard PSO	FALSE	Fully-connected	0.37	0.35	0.18	23
		Von Neumann	0.24	0.23	0.16	8
		Ring	0.29	0.29	0.16	14
	TRUE	Fully-connected	0.43	0.36	0.24	27
		Von Neumann	0.50	0.37	0.31	35
		Ring	0.42	0.34	0.25	25
Barebones PSO	FALSE	Fully-connected	0.31	0.30	0.17	15
		Von Neumann	0.26	0.27	0.16	11
		Ring	0.27	0.29	0.15	13
	TRUE	**Fully-connected**	**0.52**	**0.39**	**0.31**	**39**
		Von Neumann	0.47	0.38	0.26	33
		Ring	0.40	0.34	0.22	24
random-search	FALSE	N/A	0.30	0.30	0.17	15
	TRUE	N/A	0.47	0.38	0.24	33

Table A.10: Results for the Maze Navigation Problem using the Novelty-driven PSO, particularly, the behavior **behaviorMP1**, not archiving past behaviors. Each result represents a different experiment composed by 100 runs. $\bar{x}$, σ, $\tilde{x}$ represents, respectively, the average, standard deviation and median of the best fitnesses of each run. $\%$ success - represents the number of success runs on 100 possible.

	Always valid	Topology	$\bar{x}$	σ	$\tilde{x}$	$\%$ success
k-distances = 3	FALSE	Fully-connected	0.32	0.32	0.16	18
		Von Neumann	0.27	0.28	0.16	12
		Ring	0.28	0.27	0.17	12
	TRUE	Fully-connected	0.50	0.37	0.31	36
		Von Neumann	0.47	0.36	0.26	31
		Ring	0.52	0.37	0.31	37
k-distances = 10	FALSE	Fully-connected	0.27	0.27	0.16	11
		Von Neumann	0.28	0.26	0.17	11
		Ring	0.35	0.34	0.17	20
	TRUE	Fully-connected	0.47	0.37	0.26	31
		Von Neumann	0.51	0.38	0.31	36
		Ring	**0.58**	**0.38**	**0.33**	**45**
k-distances = 15	FALSE	Fully-connected	0.27	0.29	0.15	13
		Von Neumann	0.30	0.30	0.17	15
		Ring	0.30	0.30	0.17	15
	TRUE	Fully-connected	0.45	0.36	0.26	30
		Von Neumann	0.49	0.37	0.28	34
		Ring	0.54	0.38	0.31	40
k-distances = 25	FALSE	Fully-connected	0.22	0.24	0.15	8
		Von Neumann	0.27	0.28	0.16	12
		Ring	0.22	0.22	0.15	7
	TRUE	Fully-connected	0.39	0.33	0.24	22
		Von Neumann	0.45	0.35	0.26	29
		Ring	0.56	0.39	0.33	43

Table A.11: Results for the Maze Navigation Problem using the Novelty-driven PSO, particularly, the behavior **behaviorMP1**, archiving past behaviors. Each result represents a different experiment composed by 100 runs. $\bar{x}$, σ, $\tilde{x}$ represents, respectively, the average, standard deviation and median of the best fitnesses of each run. $\%$ success - represents the number of success runs on 100 possible.

	Always valid	Topology	$\bar{x}$	σ	$\tilde{x}$	$\%$ success
k-distances = 3	FALSE	Fully-connected	0.29	0.28	0.16	18
		Von Neumann	0.34	0.33	1.17	20
		Ring	0.29	0.28	0.17	13
	TRUE	Fully-connected	0.41	0.34	0.25	24
		Von Neumann	0.52	0.38	0.29	38
		Ring	**0.68**	**0.38**	**1.00**	**58**
k-distances = 10	FALSE	Fully-connected	0.27	0.26	0.17	11
		Von Neumann	0.28	0.28	0.17	13
		Ring	0.34	0.32	0.20	18
	TRUE	Fully-connected	0.44	0.36	0.23	29
		Von Neumann	0.58	0.39	0.33	46
		Ring	**0.68**	**0.38**	**1.00**	**58**
k-distances = 15	FALSE	Fully-connected	0.30	0.30	0.17	15
		Von Neumann	0.31	0.31	0.18	16
		Ring	0.32	0.31	0.18	17
	TRUE	Fully-connected	0.51	0.38	0.31	36
		Von Neumann	0.55	0.39	0.31	42
		Ring	0.59	0.38	0.32	46
k-distances = 25	FALSE	Fully-connected	0.33	0.33	0.17	19
		Von Neumann	0.32	0.31	0.17	17
		Ring	0.32	0.31	0.18	17
	TRUE	Fully-connected	0.49	0.38	0.26	36
		Von Neumann	0.50	0.38	0.31	36
		Ring	0.60	0.39	0.33	48

Appendix B

Results of the novelty-driven PSO and evolutionary novelty search comparison

B.1 Results of Objective-driven GE

Table B.1: Results of the objective-driven GE. Each result represents a different experiment composed by 100 runs. $\bar{x}$, σ, $\tilde{x}$ represents, respectively, the average, standard deviation and median of the best fitnesses of each run. $\%$ success - represents the number of success runs on 100 possible.

Domain	Mean best fitness	Std Deviation	Median	Successful runs
Mastermind	0.90	0.04	0.88	12
Santa Fe Trail	79.61	14.60	89.0	64
Maze problem	0.87	0.30	1.00	83

B.2 Results of Novelty-driven GE

Table B.2: Results of the novelty-driven GE applied to the Mastermind benchmark domain. Each result represents a different experiment composed by 100 runs. $\bar{x}$, σ, $\tilde{x}$ represents, respectively, the average, standard deviation and median of the best fitnesses of each run. $\%$ success - represents the number of success runs on 100 possible.

	Archived behaviors	Mean best fitness	Std Deviation	Median	Successful runs
k-distances = 3	FALSE	0.91	0.04	0.89	15
	TRUE	0.91	0.04	0.89	16
k-distances = 10	FALSE	0.91	0.04	0.89	18
	TRUE	0.91	0.04	0.89	16
k-distances = 15	FALSE	0.90	0.03	0.89	11
	TRUE	0.91	0.04	0.89	14
k-distances = 25	FALSE	0.90	0.04	0.89	12
	TRUE	0.90	0.04	0.89	20

Table B.3: Results of the novelty-driven GE applied to the Santa Fe Ant Trail benchmark domain. Each result represents a different experiment composed by 100 runs. $\bar{x}$, σ, $\tilde{x}$ represents, respectively, the average, standard deviation and median of the best fitnesses of each run. $\%$ success - represents the number of success runs on 100 possible.

	Archived behaviors	Mean best fitness	Std Deviation	Median	Successful runs
k-distances = 3	FALSE	85.89	6.39	89.0	73
	TRUE	85.33	7.75	89.0	75
k-distances = 10	FALSE	86.14	7.25	89.0	81
	TRUE	89.63	6.54	89.0	82
k-distances = 15	FALSE	87.65	4.75	89.0	88
	TRUE	87.64	4.75	89.0	90
k-distances = 25	FALSE	87.53	5.03	89.0	89
	TRUE	86.79	6.00	89.0	82

Table B.4: Results of the novelty-driven GE applied to the Maze Navigation problem. Each result represents a different experiment composed by 100 runs. $\bar{x}$, σ, $\tilde{x}$ represents, respectively, the average, standard deviation and median of the best fitnesses of each run. $\%$ success - represents the number of success runs on 100 possible.

	Archived behaviors	Mean best fitness	Std Deviation	Median	Successful runs
k-distances = 3	FALSE	1.00	0.00	1.00	100
	TRUE	1.00	0.00	1.00	100
k-distances = 10	FALSE	1.00	0.00	1.00	100
	TRUE	1.00	0.00	1.00	100
k-distances = 15	FALSE	1.00	0.00	1.00	100
	TRUE	1.00	0.00	1.00	100
k-distances = 25	FALSE	0.98	0.12	1.00	97
	TRUE	1.00	0.00	1.00	100

Abbreviations

BFN Backus Naur Form

DNA Deoxyribonucleic acid

EA Evolutionary Algorithm

EC Evolutionary Computation

EDA Estimation of Distribution Algorithms

EP Evolutionary Programming

ES Evolutionary Strategies

GA Genetic Algorithm

GP Genetic Programming

GS Grammatical Swarm

NdGS Novelty-driven Grammatical Swarm

NdPSO Novelty-driven Particle Swarm Optimization

NS Novelty Search

NT Non Terminal symbol

P Production rules

PSO Particle Swarm Optimization

RNA Ribonucleic acid

S Start symbol

T Terminal symbol

Nomenclature

$gbest$

N Total number of particles in a population.

Bibliography

[1] J Kennedy and R Eberhart. Particle swarm optimization. *Proceedings of IEEE International Conference on Neural Networks*, 4, 1995.

[2] Ender Ozcan and Chilukuri K Mohan. Analysis of a simple particle swarm optimization system. *Intelligent engineering systems through artificial neural networks*, 8:253–258, 1998.

[3] ES Peer, F Van den Bergh, and AP Engelbrecht. Using neighbourhoods with the guaranteed convergence pso. In *Swarm Intelligence Symposium, 2003. SIS'03. Proceedings of the 2003 IEEE*, pages 235–242. IEEE, 2003.

[4] Asanga Ratnaweera, Saman Halgamuge, and Harry C Watson. Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients. *Evolutionary Computation, IEEE Transactions on*, 8(3):240–255, 2004.

[5] Jakob Vesterstrom and Rene Thomsen. A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems. In *Evolutionary Computation, 2004. CEC2004. Congress on*, volume 2, pages 1980–1987. IEEE, 2004.

[6] James Kennedy. Bare bones particle swarms. In *Swarm Intelligence Symposium, 2003. SIS'03. Proceedings of the 2003 IEEE*, pages 80–87. IEEE, 2003.

[7] Joel Lehman and Kenneth O Stanley. Exploiting open-endedness to solve problems through the search for novelty. In *ALIFE*, pages 329–336, 2008.

[8] Joel Lehman, Kenneth O Stanley, and Risto Miikkulainen. Effective diversity maintenance in deceptive domains. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, pages 215–222. ACM, 2013.

[9] Joel Lehman and Kenneth O Stanley. Efficiently evolving programs through the search for novelty. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 837–844. ACM, 2010.

[10] Joel Lehman and Kenneth O Stanley. Novelty search and the problem with objectives. In *Genetic Programming Theory and Practice IX*, pages 37–56. Springer, 2011.

[11] Dario Floreano and Claudio Mattiussi. *Bio-inspired artificial intelligence: theories, methods, and technologies*. MIT press, 2008.

[12] Kenneth O Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

[13] Elena Simona Nicoară. Mechanisms to avoid the premature convergence of genetic algorithms. *Petroleum–Gas University of Ploieşti Bulletin, Math.–Info.–Phys. Series*, 61:87–96, 2009.

[14] Faustino J Gomez. Sustaining diversity using behavioral information distance. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 113–120. ACM, 2009.

[15] Paulo Urbano and Loukas Georgiou. Improving grammatical evolution in santa fe trail using novelty search. In *Advances in Artificial Life, ECAL*, volume 12, pages 917–924, 2013.

[16] Michael O'Neill and Anthony Brabazon. Grammatical swarm: The generation of programs by social programming. *Natural Computing*, 5(4):443–462, 2006.

[17] Edward O Wilson. *Sociobiology: The new synthesis*. Harvard University Press, 2000.

[18] Gerardo Beni and Jing Wang. Swarm intelligence in cellular robotic systems. In *Robots and Biological Systems: Towards a New Bionics?*, pages 703–712. Springer, 1993.

[19] Craig W Reynolds. Flocks, herds and schools: A distributed behavioral model. *ACM SIGGRAPH Computer Graphics*, 21(4):25–34, 1987.

[20] Frank Heppner and Ulf Grenander. A stochastic nonlinear model for coordinated bird flocks. *AMERICAN ASSOCIATION FOR THE ADVANCEMENT OF SCIENCE, WASHINGTON, DC(USA). 1990.*, 1990.

[21] Angelina Jane Reyes Medina, Gregorio Toscano Pulido, and José Gabriel Ramírez-Torres. A comparative study of neighborhood topologies for particle swarm optimizers. In *IJCCI*, pages 152–159, 2009.

[22] Dakuo He, Hongrui Chang, Qing Chang, and Yang Liu. Particle swarm optimization based on the initial population of clustering. In *Natural Computation (ICNC), 2010 Sixth International Conference on*, volume 5, pages 2664–2667. IEEE, 2010.

[23] Emilio F Campana, Giovanni Fasano, and Antonio Pinto. Dynamic analysis for the selection of parameters and initial population, in particle swarm optimization. *Journal of Global Optimization*, 48(3):347–397, 2010.

[24] MM Ali and P Kaelo. Improved particle swarm algorithms for global optimization. *Applied mathematics and computation*, 196(2):578–593, 2008.

[25] Yuhui Shi and Russell Eberhart. A modified particle swarm optimizer. In *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on*, pages 69–73. IEEE, 1998.

[26] Duncan J Watts. *Small worlds: the dynamics of networks between order and randomness*. Princeton university press, 1999.

[27] James Kennedy and Rui Mendes. Population structure and particle swarm performance. 2002.

[28] Bo Liu, Ling Wang, Yi-Hui Jin, Fang Tang, and De-Xian Huang. Improved particle swarm optimization combined with chaos. *Chaos, Solitons & Fractals*, 25(5):1261–1271, 2005.

[29] Jakob Vesterstrom and Rene Thomsen. A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems. In *Evolutionary Computation, 2004. CEC2004. Congress on*, volume 2, pages 1980–1987. IEEE, 2004.

[30] Jingzheng Yao and Duanfeng Han. Improved barebones particle swarm optimization with neighborhood search and its application on ship design. *Mathematical Problems in Engineering*, 2013, 2013.

[31] M Omran and S Al-Sharhan. Barebones particle swarm methods for unsupervised image classification. In *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, pages 3247–3252. IEEE, 2007.

[32] Charles Darwin. On the origins of species by means of natural selection. *London: Murray*, 1859.

[33] Michael Lynch. The frailty of adaptive hypotheses for the origins of organismal complexity. *Proceedings of the National Academy of Sciences*, 104(suppl 1):8597–8604, 2007.

[34] Daniel S Weile and Eric Michielssen. Genetic algorithm optimization applied to electromagnetics: A review. *Antennas and Propagation, IEEE Transactions on*, 45(3):343–353, 1997.

[35] Uday Kamath, Amarda Shehu, and K De Jong. Using evolutionary computation to improve svm classification. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1–8. IEEE, 2010.

[36] Gregory Hornby, J Lohn, and D Linden. Computer-automated evolution of an x-band antenna for nasa's space technology 5 mission. *Evolutionary computation*, 19(1):1–23, 2011.

[37] Hans-Paul Schwefel. Kybernetische evolution als strategie der experimentellen forschung in der strömungstechnik. *Master's thesis, Hermann Föttinger Institute for Hydrodynamics, Technical University of Berlin*, 1965.

[38] Lawrence J Fogel, Alvin J Owens, and Michael J Walsh. Artificial intelligence through simulated evolution. 1966.

[39] John H Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence.* U Michigan Press, 1975.

[40] John R Koza. *Genetic programming: on the programming of computers by means of natural selection*, volume 1. MIT press, 1992.

[41] Adam Lipowski and Dorota Lipowska. Roulette-wheel selection via stochastic acceptance. *Physica A: Statistical Mechanics and its Applications*, 391(6):2193–2196, 2012.

[42] L Darrell Whitley et al. The genitor algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. In *ICGA*, volume 89, pages 116–123, 1989.

[43] Brad L Miller and David E Goldberg. Genetic algorithms, tournament selection, and the effects of noise. *Complex Systems*, 9(3):193–212, 1995.

[44] JORGE Magalhães-Mendes. A comparative study of crossover operators for genetic algorithms to solve the job shop scheduling problem. *WSEAS transactions on computers*, 12(4), 2013.

[45] Thomas Jansen and Ingo Wegener. The analysis of evolutionary algorithms-a proof that crossover really can help. *Algorithmica*, 34(1):47–66, 2002.

[46] Zbigniew Michalewicz. *Genetic algorithms+ data structures= evolution programs*. Springer Science & Business Media, 1996.

[47] Conor Ryan, JJ Collins, and Michael O Neill. Grammatical evolution: Evolving programs for an arbitrary language. In *Genetic Programming*, pages 83–96. Springer, 1998.

[48] Anthony Brabazon and Michael O'Neill. *Biologically inspired algorithms for financial modelling*. Springer Science & Business Media, 2006.

[49] Jason H Moore and Lance W Hahn. Systems biology modeling in human genetics using petri nets and grammatical evolution. In *Genetic and Evolutionary Computation–GECCO 2004*, pages 392–401. Springer, 2004.

[50] Martin Hemberg and Una-May O'Reilly. Genr8-using grammatical evolution in a surface design tool. In *GECCO*, pages 120–123, 2002.

[51] Gregor Mendel. Versuche über pflanzenhybriden. *Verhandlungen des naturforschenden Vereines in Brunn 4: 3*, 44, 1866.

[52] Joel Lehman and Kenneth O Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary computation*, 19(2):189–223, 2011.

[53] Jorge Gomes, Paulo Urbano, and Anders Lyhne Christensen. Introducing novelty search in evolutionary swarm robotics. In *Swarm Intelligence*, pages 85–96. Springer, 2012.

[54] David E Goldberg. Simple genetic algorithms and the minimal, deceptive problem. *Genetic algorithms and simulated annealing*, 74:88, 1987.

[55] Martin Pelikan and David E Goldberg. Escaping hierarchical traps with competent genetic algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 511–518, 2001.

[56] Gunar E Liepins Michael D Vose. Deceptiveness and genetic algorithm dynamics'. *Foundations of Genetic Algorithms 1991 (FOGA 1)*, 1:36, 2014.

[57] Jianjun Hu, Erik Goodman, Kisung Seo, Zhun Fan, and Rondal Rosenberg. The hierarchical fair competition (hfc) framework for sustainable evolutionary algorithms. *Evolutionary Computation*, 13(2):241–277, 2005.

[58] Gregory S Hornby. Alps: the age-layered population structure for reducing the problem of premature convergence. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 815–822. ACM, 2006.

[59] David E Goldberg and Jon Richardson. Genetic algorithms with sharing for multimodal function optimization. In *Genetic algorithms and their applications: Proceedings of the Second International Conference on Genetic Algorithms*, pages 41–49. Hillsdale, NJ: Lawrence Erlbaum, 1987.

[60] Marcus Hutter. Fitness uniform selection to preserve genetic diversity. In *Evolutionary Computation, 2002. CEC'02. Proceedings of the 2002 Congress on*, volume 1, pages 783–788. IEEE, 2002.

[61] David E Goldberg, Bradley Korb, and Kalyanmoy Deb. Messy genetic algorithms: Motivation, analysis, and first results. *Complex systems*, 3(5):493–530, 1989.

[62] Jeffrey L Elman. *Incremental learning, or the importance of starting small*. University of California, San Diego, 1991.

[63] Faustino Gomez and Risto Miikkulainen. Incremental evolution of complex general behavior. *Adaptive Behavior*, 5(3-4):317–342, 1997.

[64] Stéphane Doncieux and J-B Mouret. Behavioral diversity measures for evolutionary robotics. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1–8. IEEE, 2010.

[65] John Doucette and Malcolm I Heywood. Novelty-based fitness: An evaluation under the santa fe trail. In *Genetic Programming*, pages 50–61. Springer, 2010.

[66] Enrique Naredo, Leonardo Trujillo, and Yuliana Martínez. *Searching for novel classifiers*. Springer, 2013.

[67] Uri Wilensky et al. Netlogo. evanston, il. *Center for Connected Learning and Computer Based Modeling, Northwestern University. http://ccl. northwestern. edu/netlogo*, 1999.

[68] Loukas Georgiou and William J Teahan. jge-a java implementation of grammatical evolution. In *Proceedings of the 10th WSEAS International Conference on SYSTEMS, Vouliagmeni. Athens: WSEAS*, pages 406–411, 2006.

[69] Loukas Georgiou and William J Teahan. Grammatical evolution and the santa fe trail problem. *Evolutionary Computation (ICEC 2010)*, 2010.

[70] John R Koza. Genetic evolution and co-evolution of computer programs. *Artificial life II*, 10:603–629, 1991.

[71] Loukas Georgiou. *Constituent grammatical evolution*. PhD thesis, Bangor University, 2012.

[72] XH Shi, YC Liang, HP Lee, C Lu, and LM Wang. An improved ga and a novel pso-ga-based hybrid algorithm. *Information Processing Letters*, 93(5):255–261, 2005.

[73] K Premalatha and AM Natarajan. Hybrid pso and ga for global maximization. *Int. J. Open Problems Compt. Math*, 2(4):597–608, 2009.

[74] Maarten Keijzer, Michael O'Neill, Conor Ryan, and Mike Cattolico. Grammatical evolution rules: The mod and the bucket rule. In *Genetic Programming*, pages 123–130. Springer, 2002.

[75] Ann Thorhauer and Franz Rothlauf. On the locality of standard search operators in grammatical evolution. In *Parallel Problem Solving from Nature–PPSN XIII*, pages 465–475. Springer, 2014.

[76] Nuno Lourenço, Francisco B Pereira, and Ernesto Costa. Studying the properties of structured grammatical evolution. 2015.

Index